Philipps Universität Marburg

Raster Time Series: Learning and Processing

Dissertation

zur Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

M.Sc. Geographie
B.Sc. Informatik
Johannes Drönner
geboren in Marburg

Marburg, im August 2019

Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg
(Hochschulkennziffer 1180) als Dissertation am 12. September 2019 angenommen.

Erstgutachter: Prof. Dr. Bernhard Seeger, Philipps-Universität Marburg

Zweitgutachter: Prof. Dr. Jörg Bendix, Philipps-Universität Marburg

Tag der Einreichung: 20. August 2019

Tag der mündlichen Prüfung: 20. September 2019

—Dedicated to all the Meteosat Second Generation images out there

Abstract

As the amount of remote sensing data is increasing at a high rate, due to great improvements in sensor technology, efficient processing capabilities are of utmost importance. Remote sensing data from satellites is crucial in many scientific domains, like biodiversity and climate research. Because weather and climate are of particular interest for almost all living organisms on earth, the efficient classification of clouds is one of the most important problems. Geostationary satellites such as Meteosat Second Generation (MSG) offer the only possibility to generate long-term cloud data sets with high spatial and temporal resolution. This work, therefore, addresses research problems on efficient and parallel processing of MSG data to enable new applications and insights.

First, we address the lack of a suitable processing chain to generate a long-term Fog and Low Stratus (FLS) time series. We present an efficient MSG data processing chain that processes multiple tasks simultaneously, and raster data in parallel using the Open Computing Language (OpenCL). The processing chain delivers a uniform FLS classification that combines day and night approaches in a single method. As a result, it is possible to calculate a year of FLS rasters quite easy.

The second topic presents the application of Convolutional Neural Networks (CNN) for cloud classification. Conventional approaches to cloud detection often only classify single pixels and ignore the fact that clouds are highly dynamic and spatially continuous entities. Therefore, we propose a new method based on deep learning. Using a CNN image segmentation architecture, the presented Cloud Segmentation CNN (CS-CNN) classifies all pixels of a scene simultaneously. We show that CS-CNN is capable of processing multispectral satellite data to identify continuous phenomena such as highly dynamic clouds. The proposed approach provides excellent results on MSG satellite data in terms of quality, robustness, and runtime, in comparison to Random Forest (RF), another widely used machine learning method.

Finally, we present the processing of raster time series with a system for Visualization, Transformation, and Analysis (VAT) of spatio-temporal data. It enables data-driven research with explorative workflows and uses time as an integral dimension. The combination of various raster and vector data time series enables new applications and insights. We present an application that combines weather information and aircraft trajectories to identify patterns in bad weather situations.

Zusammenfassung

Da die Menge der Fernerkundungsdaten stark zunimmt, sind effiziente Verarbeitungsmöglichkeiten von größter Bedeutung. Fernerkundungsdaten von Satelliten sind in vielen wissenschaftlichen Bereichen, wie Biodiversität und Klimaforschung, von entscheidender Bedeutung. Wetter und Klima für fast alle lebenden Organismen auf der Erde von besonderem Interesse. Daher ist die effiziente Klassifizierung von Wolken eines der wichtigsten Probleme. Geostationäre Satelliten, wie Meteosat Second Generation (MSG), bieten die einzige Möglichkeit, langfristige Wolkendatensätze mit hoher räumlicher und zeitlicher Auflösung zu erzeugen. Diese Arbeit befasst sich daher mit der effizienten und parallelen Verarbeitung von MSG-Daten, um neue Anwendungen und Erkenntnisse zu ermöglichen.

Zuerst wird das Fehlen einer geeigneten Verarbeitungskette zur Erzeugung einer langfristigen Nebel- und niedrgem Stratus- (FLS) Zeitreihe aufgegriffen. Wir präsentieren eine effiziente MSG-Datenverarbeitungskette, die mehrere Aufgaben gleichzeitig, sowie Rasterdaten parallel, mit der Open Computing Language, verarbeitet. Die Verarbeitungskette ermöglicht eine einheitliche Klassifizierung von FLS, die Tag- und Nachtansätze kombiniert. Infolgedessen ist es möglich, die FLS-Raster für ein Jahr in kurzer Zeit zu berechnen.

Das zweite Thema präsentiert die Anwendung von Convolutional Neural Networks (CNN) zur Wolkenklassifikation. Gängige Ansätze zur Wolkenerkennung klassifizieren oft nur einzelne Pixel und ignorieren die Tatsache, dass Wolken hochdynamische und räumlich kontinuierliche Einheiten sind. Daher schlagen wir eine neue Methode vor, die auf Deep Learning basiert. Basierend auf einer CNN-Architektur zur Bildsegmentierung klassifiziert das vorgestellte Cloud Segmentation CNN (CS-CNN) alle Pixel einer Szene gleichzeitig. Wir zeigen, dass CS-CNN multispektrale Satellitendaten verarbeiten kann, um kontinuierliche Phänomene, wie Wolken zu identifizieren. Unser Ansatz liefert ausgezeichnete Ergebnisse auf MSG-Satellitendaten in Bezug auf Qualität, Robustheit und Laufzeit im Vergleich zu Random Forest, einer oft verwendeten maschinellen Lernmethode.

Schließlich stellen wir die Verarbeitung von Rasterzeitreihen mit einem System zur Visualisierung, Transformation und Analyse (VAT) von raumzeitlichen Daten vor. Es ermöglicht daten-getriebene Forschung mit explorativen Workflows und verwendet Zeit als integralen Bestandteil. Die Kombination aus heterogenen Raster- und Vektordaten-Zeitreihen ermöglicht neue Anwendungen und Erkenntnisse. Wir stellen einen Anwendungsfall vor, der Wetterinformationen und Flugzeugtrajektorien kombiniert, um Muster in Schlechtwettersituationen zu identifizieren.

Erklärung

Hiermit versichere ich, dass ich meine Dissertation mit dem Titel

Raster Time Series: Learning and Processing

selbständig und ohne fremde Hilfe verfasst, nicht andere als die in ihr angegebenen Quellen oder Hilfsmittel benutzt, alle vollständig oder sinngemäß übernommenen Zitate als solche gekennzeichnet sowie die Dissertation in der vorliegenden oder einer ähnlichen Form noch bei keiner anderen in- oder ausländischen Hochschule anlässlich eines Promotionsgesuchs oder zu anderen Prüfungszwecken eingereicht habe. Dies ist mein erster Versuch einer Promotion.

Marburg, den 20. August 2019 Johannes Drönner

Acknowledgments

Without the support of my colleagues, friends, and family, it would not have been possible to finish this work successfully.

First of all, I would like to thank Prof. Dr. Bernhard Seeger, who supervised me through the course of this thesis. He not only provided the financial and organizational framework for this project but also granted me the freedom to develop my ideas and gave me orientation and advise when I struggled.

Because I know how vital a positive working environment is, I want to thank all my colleagues from the Database Research Group of the University of Marburg for the good teamwork, interesting discussions, and exciting projects. In particular, I would like to mention Christian Beilschmidt and Michael Mattig. Many thanks to you for the excellent team, the very valuable support of this thesis, and all the fun, even if it wasn't that easy from time to time. Furthermore, I want to thank Mechthild Keßler for supporting me in all organizational aspects.

I also want to thank Markus Mühling and Nikolaus Korfhage for sharing profound knowledge about deep learning and the cooperation, which resulted in the successful cloud segmentation.

Many thanks also to my second home, the Department of Geography at the University of Marburg. I would especially like to thank Prof. Dr. Jörg Bendix and Boris Thies for the many years of support. I am also very grateful to Hanna Meyer and Sebastian Egli, whom I was always allowed to visit unannounced to discuss clouds and other topics.

My work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) under grant no. SE 553/7-1 and SE 553/7-2. As part of the GFBio project, I am particularly thankful for the provision of use cases of biodiversity research that lead to the development of the Visualization, Analysis and Transformation System. I also want to thank the people at the European Organisation for the Exploitation of Meteorological Satellites for providing the satellite data that forms the basis of my work.

Finally, I would also like to thank my family and friends who have accompanied me during this time and especially supported me in the not so uncomplicated situations. Thank you all for giving me a supportive background that made it possible for me to get to this point. In particular, I would like to thank Jana for proofreading this thesis and for her motivation and support.

Contents

1 Introduction

Remote sensing data from satellites play a crucial role in many domains, like biodiversity and climate research. The arrival of more and more earth observation satellites will increase the amount of available data and the need for efficient data handling [Yan+17; Ove+11]. In particular, reliable information about clouds is of utmost importance in various applications. Clouds are essential for our climate and influence many aspects of life on Earth. Remote sensing techniques using satellites provide the only way to generate global and long-term cloud datasets with spatial and temporal resolutions required for a plethora of applications [Stu+13].

Clouds can mask study areas, and thus make it more challenging to gain knowledge, e.g., for the estimation of forest cover [Han+08]. However, clouds are also study-objects themselves. Thies and Bendix [TB11] give an excellent overview of cloud detection methods. Obviously, clouds are indicators for global and local weather conditions, e.g., for fog [ETB18; Gul+07] or rainfall [Mey+16]. They occur in extreme weather events like storms and heavy rainfall that can cause severe damages and threaten human life [Hou14a]. The high dynamic of clouds is emphasized in detail by Houze [Hou14b]. Products derived from satellite data enable monitoring, prediction, and consequently preparation and reaction to global as well as local weather and climate situations.

Clouds reduce solar irradiation and therefore limit solar power production. This dynamic effect can destabilize energy grids [Köh+17]. Tracking of biodiversity also relies on satellite data. In biodiversity and other domains, clouds are either indicators [WJ16] or obstacles to answering research questions, causing the necessity of cloud-free images [Han+08]. The availability of time series is another aspect, which is often only provided by long-term satellite missions. Norris *et al.* [Nor+16] report that long-term satellite data indicate evidence for climate change.

Clouds appear in very different forms and sizes. One of them is Fog and Low Stratus (FLS), which provides the main water supply for some, otherwise arid,

ecosystems [BEB05; VB99]. In contrast to this, FLS have critical downsides. Inversion situations can lead to smog, with negative effects on human and animal health [NHN01]. The most direct impact of FLS concerns land, air, and sea traffic, where reduced visibility leads to hazardous situations with more frequent and severe accidents [BEK11].

1.1 Motivations and Hypothesis

This thesis presents different approaches for processing, classification, and interactive analysis of spatio-temporal raster data. The following subsections identify deficits and our aims and define the working hypothesis for this thesis. The first aim is tackling the parallel and efficient processing of satellite raster data. The second part extends the developed processing chain for homogeneous FLS detection. Furthermore, we investigate the possibilities of modern Convolutional Neural Networks (CNN) for instant cloud classification. In order to explore spatio-temporal data interactively, a web-based Geographic Information System (GIS) is presented with examples for raster time series.

1.1.1 Meteosat Second Generation Processing

Several approaches [Ban+09; TB11; TC13] have been developed to classify pixels of satellite images as clouded or cloud-free as well as to detect more complex cloud classes like FLS [BB91; Mus+14; Sch+16; CB07].

Low Earth Orbiters (LEO) provide data with a high spatial resolution. However, the revisit time of LEOs is in the order of days. GEOstationary satellites (GEO), on the other hand, provide raster data with very high temporal resolution. The temporal resolution, depending on the repeat cycle of the spacecraft, renders LEO-based data problematic for real-time monitoring and coverage of diurnal dynamics. GEOs, e.g., Meteosat Second Generation (MSG), provide the required resolutions for nowcasting and analytics of long time-series. MSG has a spatial resolution of 3 km×3 km and a temporal resolution of 15 min. It provides data from 11 primary channels, which are sensitive to reflected sunlight and infrared emission from the earth's surface [Sch+02]. MSG is operating since 2004 and produces 35,040 scenes each year. Meteosat Third Generation (MTG) will offer an improved spatial resolution and cover more bands [Stu+05]. For regions of interest, new satellites, like the Japanese GEO Himawari 8/9, provide spatial resolutions of 0.5–2 km per pixel and temporal resolutions of 2.5 min [Bes+16]. Therefore, algorithms

must be able to produce results within very short image repetition cycles and for high spatial resolutions.

To handle the produced remote sensing big data [Ma+15], efficient algorithms, and fast processing are required. Rule-based classification of remote sensing raster data is often re-implemented for timeliness [HFS10] or run with reduced resolutions to allow nowcasting [NWC13]. Climate science requires long-time series, and nowcasting requires timely results, e.g., to issue warnings or to reroute traffic in FLS situations. Therefore, fast processing is required for real-time information as well as for long time series. It is also necessary to enable recalculation and ad-hoc computation as well as exploratory parameter changes, dynamic study areas, and evaluation of new methods. Therefore, all techniques presented in this work are developed with fast and parallel processing capabilities as a primary goal.

New development of products, e.g., for cloud and FLS detection, as well as nowcasting and other applications, require fast and parallel processing methods for MSG data. The development of an extensible processing chain for efficient and parallel MSG raster data processing was, therefore, the first **aim** of this thesis. The first working Hypothesis **(H1)** is formulated as follows:

H1 Efficient, parallel, and concurrent methods enable processing of large MSG time series with very low processing times and enable the development of new applications.

1.1.2 Fog and Low Stratus Detection

While cloud time series of ten years or more exist, e.g., Benas *et al.* [Ben+17], there is no long-term FLS dataset with high spatial and temporal resolutions. Schemes to derive FLS products from MSG data have been developed for night [CB08] and day [CB07]. While initially developed for a study-area covering Europe, they have been adapted to other study-areas [Cer12]. However, the design of the methods requires a specific adaption to each fix sized study area. Handling of dynamic study areas, as well as night and day collectively, requires a more flexible implementation. Using both schemes separately, Cermak *et al.* [Cer+09] created a day and night time series for the winter months (Dec, Jan, Feb) from 2004 to 2008. However, there is no long-term, e.g., ten years, FLS time series covering the diurnal cycle with one homogeneous classification approach.

To resolve both **deficits**, the dependency on fixed study areas and lack of a long-term diurnal FLS dataset, a unified FLS classification is presented. The day and

night schemes [CB07; CB08] are unified to use consistent partitioning strategies and enhanced for faster processing. Our approach supports flexible study areas of arbitrary size. The modular implementation is building on the developed MSG processing chain, which uses efficient algorithms, optimized loading strategies, and parallelism.

The second working hypothesis of this thesis is, therefore, formulated as follows.

H2 A unified and area independent spatial partitioning strategy enables homogeneous FLS classification for large time series and parallel processing.

1.1.3 Cloud Classification with Convolutional Neural Networks

Methods for cloud detection and classification range from rule sets to sophisticated methods from the machine learning domain. Rules and thresholds, as well as machine learning models, allow classifying single pixels as clouded or cloud-free. While classification can also target multiple classes and detect entities, the focus is frequently on methods applied to individual pixels.

Classification of cloud pixels with rules or machine learning techniques, e.g., Random Forst (RF) [ETB18] or Neural Networks (NN) [Mey+16], have two significant **deficits**. First, most of them focus on the independent classification of individual pixels and rarely use spatial information that extends beyond adjacent pixels. Thus, these pixel-based approaches ignore the fact that clouds are a spatially continuous and highly dynamic phenomenon. Manually creation of spatial statistics about the closer environment (e.g., 3×3 pixel) of individual pixels covers only a part of the available spatial information and ignores the spatial relationships and dynamics in broader areas and scales. Second, domain experts must manually create and select the most appropriate features for the classification task. This is a quite cumbersome and time-consuming task due to a massive number of possible features. It is therefore not feasible to create all features manually for multi-spectral raster data in the terabyte range. At the same time, feature selection also reduces potential training data.

Deep learning approaches have shown great promises in a broad spectrum of applications [LBH15]. In particular, Convolutional Neural Networks (CNN) [KSH12; LSD15; He+17] are highly suitable for object detection, image classification, and segmentation. Unlike other methods, CNNs automatically learn the most important features without involving domain experts to create or select them manually. Despite their advantages, all the existing methods for cloud classification follow the

approach to treat all pixels individually when the model is trained and later when the trained model is applied. However, CNN architectures for image segmentation [RPB15] holistically classify an entire image.

This thesis presents a cloud classification approach based on a CNN architecture for image segmentation, which requires no hand-crafted features and no feature selection. Additionally, the architecture is adapted to multispectral satellite data. It also provides a instant classification of clouds. To study this, two **work packages** are derived. The first one aims to generate the required data as well as training and evaluation of a state-of-the-art machine learning method as a competitor for a CNN approach. The second aims at the the design, training, and evaluation of the Cloud Segmentation CNN (CS-CNN). Therefore, the third working hypothesis is formulated as follows.

H3 CNNs enable instant cloud classification without feature engineering and selection by domain experts and include large scale spatial context.

1.1.4 Visualization and Explorative Analysis

Batch processing of spatio-temporal raster data is one important aspect. However, access and visualization are another. The combination of remote sensing and other spatio-temporal data can provide new insights and scientific ideas, in biodiversity research, the combination of satellite data and trajectories facilitates new insights. While analysis often includes terrain data, the incorporation of time series with weather information is more complex [Kay+15]. Kuenzer *et al.* [Kue+14] report a multitude of limitations as the reason for this. Large raster datasets are not easy to access and processing requires domain expert knowledge. An interactive system, which provides easy access to spatio-temporal data and integrates data handling, is essential for many domains [Gil+18]. Existing web-based systems have two essential deficits. First, most systems lack support for time as an integral dimension. Therefore, users have to to work with single time snapshots instead of time series. Second, web-based systems only provide a limited set of options for combining data and analysis.

The Database Research Group at the University of Marburg develops the Visualization, Analysis, and Transformation (VAT) System [Bei+17b]. It combines web-based visualizations of spatio-temporal big data with interactive analytics. Thus, it enables data-driven research using the concept of exploitative workflows. While most Geographic Information Systems (GIS) do not handle time as integral domain, VAT uses a workflow approach to apply user-defined analysis to arbitrary steps in time. VAT also exceeds the functionality of web-based special-purpose

applications, e.g., Map of Life [JMG12], which provide time selection but only a limited set of analytic operations. This thesis presents examples of raster time series applications and processing capabilities in VAT. Different use-cases introduce the processing back-end and the interactive front-end. We aim to support data exploration and visual analytics on spatio-temporal data to enable new insights through combinations of remote sensing raster and event data, e.g., changing positions of animals or planes.

Therefore, the fourth working hypothesis is formulated as follows.

H4 Interactive explorations and analytics for spatio-temporal data enable insights beyond pure event or raster processing.

1.2 Publications

The following papers were published in the course of this thesis:

- Johannes Drönner, Sebastian Egli, Boris Thies, Jörg Bendix, Bernhard Seeger:
  FFLSD - Fast Fog and Low Stratus Detection tool for large satellite time-series.
  Computers & Geosciences 128: 51-59 (2019).
- Johannes Drönner, Nikolaus Korfhage, Sebastian Egli, Markus Mühling, Boris Thies, Jörg Bendix, Bernd Freisleben, Bernhard Seeger:
  Fast Cloud Segmentation Using Convolutional Neural Networks.
  Remote Sensing 10(11): 1782 (2018).
- Christian Beilschmidt, Johannes Drönner, Nikolaus Glombiewski, Christian Heigele, Jana Holznigenkemper, Anna Isenberg, Michael Körber, Michael Mattig, Andreas Morgen, Bernhard Seeger:
  Pretty Fly for a VAT GUI: Visualizing Event Patterns for Flight Data.
  DEBS 2019: 224-227.
- Christian Beilschmidt, Johannes Drönner, Michael Mattig, Marco Schmidt, Christian Authmann, Aidin Niamir, Thomas Hickler, Bernhard Seeger:
  VAT: A Scientific Toolbox for Interactive Geodata Exploration.
  Datenbank-Spektrum 17(3): 233-243 (2017).
- Christian Beilschmidt, Johannes Drönner, Michael Mattig, Bernhard Seeger:
  VAT: A System for Data-Driven Biodiversity Research.
  EDBT 2017: 546-549.
- Christian Beilschmidt, Johannes Drönner, Michael Mattig, Marco Schmidt, Christian Authmann, Aidin Niamir, Thomas Hickler, Bernhard Seeger:
  Interactive Data Exploration for Geoscience.
  BTW Workshops 2017: 117-126.
- Hanna Meyer, Johannes Drönner, Thomas Nauss:
  Satellite-based high-resolution mapping of rainfall over southern Africa.
  Atmospheric Measurement Techniques 10 (6): 2009-2019.
- Sebastian Egli, Boris Thies, Johannes Drönner, Jan Cermak, Jörg Bendix:
  A 10 year fog and low stratus climatology for Europe based on Meteosat Second Generation data.
  Quarterly Journal of the Royal Meteorological Society 143 (702), 530-541.

- Christian Authmann, Christian Beilschmidt, Johannes Drönner, Michael Mattig, Bernhard Seeger: **VAT: A System for Visualizing, Analyzing and Transforming Spatial Data in Science.**
  Datenbank-Spektrum 15(3): 175-184 (2015).
- Christian Authmann, Christian Beilschmidt, Johannes Drönner, Michael Mattig, Bernhard Seeger:
  Rethinking Spatial Processing in Data-Intensive Science.
  BTW Workshops 2015: 161-170.

1.3 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2 presents an overview of related topics and essential concepts. This covers the introduction of spatio-temporal data types and operations on raster data. It includes the basics of geographical information systems, remote sensing, GPU processing, and machine learning. A focus is on the raster time series from the European weather satellite Meteosat Second Generation (MSG) and a derived cloud mask. Random Forest (RF) and Convolutional Neural Networks (CNN) are discussed in particular.

This chapter contains an extended description of the data from [Drö+18] and [Drö+19].

Chapter 3 presents related work. Methods and systems implementing efficient and parallel raster data processing are presented first. Then, related work for cloud and fog detection in context of remote sensing is presented. With a focus on this topic, machine learning applications like RF and CNNs are presented. The chapter concludes with related work on interactive visualization and analysis of spatio-temporal data.

This chapter contains parts of [Drö+18] and [Drö+19].

Chapter 4 presents an efficient processing chain for loading and calibrating MSG raster data. Efficient algorithms, GPU processing and concurrent execution of all processing steps enable very short runtimes. Experiments show the impact of GPUs and concurrent execution.

This chapter contains parts of [Drö+19].

Chapter 5 builds on the processing chain from Chapter 4 to generate large Fog and Low Stratus (FLS) time series. A unification of day and night methods for FLS detection and techniques for efficient and concurrent processing are presented. Experiments show the impact of concurrent and efficient implementations.

This chapter contains parts of [Drö+19] and [Egl+17].

Chapter 6 presents a straight forward adaption of RF for cloud pixel classification in MSG data. Important features for could classification are spatial statistics, which introduce higher level information into a model. A comparison between two RF models with and without spatial statistics provides evidence. The evaluation covers single scenes as well as diurnal and seasonal effects.

This chapter contains parts of [Drö+18].

Chapter 7 introduces a novel method for cloud mask creation using CNNs. The presented approach uses a CNN to classify all pixels simultaneously with a technique called segmentation. The Cloud Segmentation CNN (CS-CNN) is trained with six years of MSG data and processes a single raster within 25 milliseconds. CS-CNN outperforms the RF methods from Chapter 6 as a detailed evaluation shows.

This chapter contains parts of [Drö+18].

Chapter 8 switches from processing time series with focus on throughput and high speed for full resolutions to interactive visualization. The Visualization, Analysis, and Transformation System (VAT) enables explorative workflows for investigation and combination of spatio-temporal data. A use-case shows the combination of cloud data and trajectories of planes.

This chapter contains parts of [Bei+17b] and [Bei+19a].

Chapter 9 revisits the hypothesis and evaluates them. It concludes the thesis and presents ideas for future work.

2
Fundamentals

This chapter presents the concepts of spatio-temporal data with a focus on raster time series. First, spatio-temporal data and the raster and vector models are introduced. Second, we give a brief overview of Geographical Information Systems (GIS) and raster operations. Then, remote sensing and the properties of spatial, temporal, and spectral resolutions are presented. We discuss the datasets used throughout this work in detail. Furthermore, examples of applications or operations are given. Processing on Graphics Processing Units (GPU) and Machine Learning (ML) is a focus of this work. Therefore, the basics of the Open Compute Language (OpenCL) as well as two ML methods, Random Forest (RF) and Convolutional Neural Networks (CNN) are presented.

2.1 Spatio-Temporal Data

Spatio-temporal data have, as the name suggests, two components. First of all, they are spatially located by coordinates, which represent points on Earth (or generally in space). Furthermore, these data objects consist of a time component, which indicates when or in which period the observed event happened.

First, we will look at the spatial component. Spatial objects or phenomena are represented as raster or vector data. Figure 2.1 shows a polygon as an example for vector data on the left side. Vector data ultimately consist of one or many points with coordinates. Polygons, like the one in the example, are defined by a ring of points. The vector data model is an object-based model. An object can contain several attributes like the name of the covered area, an id, or the area size. On the right side, the figure shows three raster layers. Raster layers are composed of a grid of pixels, where each pixel can have an individual value. This is required to represent the temperature or the elevation for the earth's surface. Vector data represents discrete information, and raster usually represents continuous phenomena,

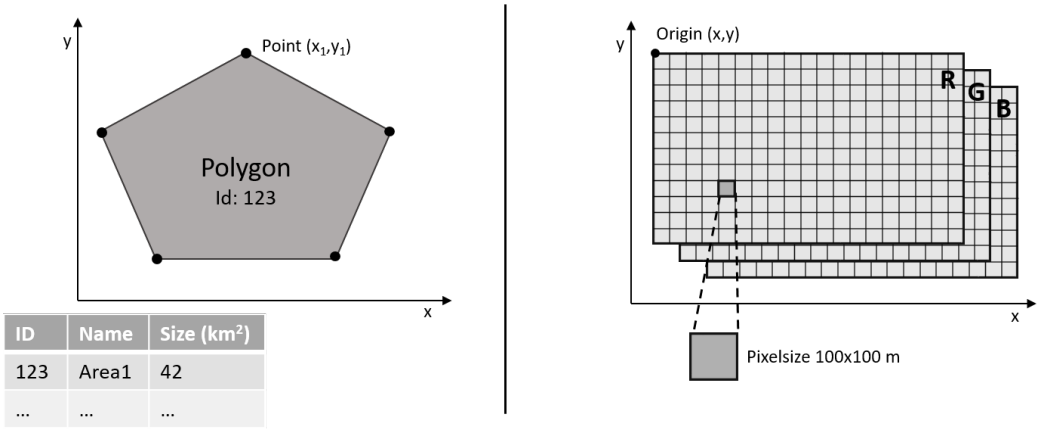


Figure 2.1: A visualization of the vector (left) and the raster model (right). The vector model contains discrete objects, which are composed of points with spatial coordinates. A vector object can have multiple attributes. The raster model represents continuous phenomena in a grid of pixels. Each raster layer presents only one phenomenon. RGB images, therefore, require three layers.

e.g., surface temperatures. While vector objects can have multiple attributes, a single raster can represent only one phenomenon.

The Simple Feature Model [Ope10a] defines vector data based on points with two dimensional or three-dimensional coordinates. The concatenation of points forms lines while polygons have an outer ring and a set of inner rings representing holes in the polygon. Coordinates of points are defined in the Cartesian or polar coordinate system, e.g., Latitude (Lat) and Longitude (Lon) and optional height. Examples of points are the locations of trees, while lines allow representing the course of rivers or roads. Polygons are used to model discrete areas like countries, houses, or habitats. A geometry, such as a point, line or polygon in combination with one or multiple attributes, is referred to as feature. A set of vector objects is usually modeled in a relational fashion, where each object represents one record.

The raster model is much simpler than the vector model. In contrast to the object-based model of vector data, raster data follows a layer model. As indicated in Figure 2.1, a single raster layer can represent only a single but continuous phenomenon.Defining the spatial properties of a raster requires the coordinate of its origin in a Cartesian coordinate system, as indicated in Figure 2.1. Additionally, the pixel size is defined for each dimension, e.g., 100 m×100 m. In a raster, pixels are addressed by row and column numbers. The spatial position of each pixel is

determined by the origin, the pixel size, and its row and column number [De 13]. Examples for raster data are elevation maps or surface temperatures. Satellite images, in this case, photos of the earth from space, require three raster layers to store the red, green, and blue (RGB) information. A comparison of vector and raster data model [De 13] shows that both are required for many applications and analyses as both fulfill different requirements. The raster format is the natural one for remote sensing [RR99].

Both types of data are transformable into each other. However, this transformation can cause a loss of accuracy. Rasterization transforms vector data into raster layers, where each attribute requires an individual layer. The other way around, vectorization transforms raster data into vectors. When transforming a raster pixel into the vector model, usually the coordinates of pixel centers are selected to represent the location of the object [De 13].

The Simple Feature Model does not define the temporal aspect of spatial vector data. However, spatial data can change over time. For example, rain gauges measure different amounts of precipitation at different times. In this case, only the attributes of the objects change while the locations are fixed. There is also the case that objects move to other locations [De 13]. Examples for moving objects [Güt+00] are airplanes and animals. As the Simple Feature Model does not model time separately, it is usually represented as an attribute per feature. Spatio-temporal vector data is also considered in event processing. Events are defined as objects with time intervals in which they are valid. For event data, it is possible to determine a snapshot of only valid data at any given point in time [KS05].

GIS also use a snapshot approach to handle the temporal component of spatial data [Peu99]. The raster data model limits the observed phenomenon to one attribute at a single time interval. Therefore, a single raster layer can only represent one time step and a sequence of snapshots models a time series. This is particularly relevant for remote sensing where satellites continuously generate new raster data. When using satellites for remote sensing, the time between two scans of the same area defines the temporal resolution [RR99].

2.2 GIS and Raster Processing

Geographical Information Systems (GIS) allow to manage, transform, and visualize spatial data. They also provide different tools for analysis [SW04; HdB09]. There are many kinds of GIS with different focus and types of user interaction. Desktop

software like ESRI ArcGIS[1], QGIS[2], GRASS[3], or Google Earth[4] provide visual user interfaces and interactive toolboxes. Spatial database systems provide data management and allow users to use declarative queries on vector or raster data. However, spatial databases often support only one of the spatial data types very well. PostGIS [OH15] has a focus on vector data and Rasdaman [Bau99] is specialized on spatio-temporal raster data. Another example is the spatio-temporal database Secondo [Güt+05], which stands out with a focus on support for moving objects. While the purpose of GIS software varies, data access and basic operations are common.

2.2.1 Data access

The Geospatial Data Abstraction Library[5] (GDAL) is a data access layer for spatial data. It is frequently used in GIS and by data scientists, which use scripting languages like R[6] or Python[7]. GDAL provides access to many different data formats and wraps them into homogeneous data models. While the implementation of the raster model is simply referred to as GDAL, the implementation of the vector model is called Open GIS Simple Feature Reference (OGR). According to the Simple Feature Model, features are objects with geometries and multiple attributes. OGR also provides a Common Query Language (CQL) with spatial operations and attribute filters. For raster data, the abstraction allows to address layers (called bands) as arrays and provides information like origin and pixel size. GDAL also provides access to raster dataset and raster layer metadata. Temporal information is either stored as attributes for individual vector objects, or as metadata for raster layers. However, since there is no standard, users and data providers often use the file name to represent the snapshot time of the contained data.

Remote sensing data from satellites, drones, or airplanes, as well as weather or climate models, provide large high-resolution time series [Yan+17]. The size of spatio-temporal datasets can easily exceed the range of terabytes (TB). For efficient access to raster data, GDAL and many GIS employ two techniques called *tiling* and *pyramids*. A pyramid represents a hierarchy of layers generated by applying an aggregation function to a raster layer in a recursive fashion dividing the size by two

[1] esri.com/arcgis
[2] qgis.org
[3] grass.osgeo.org
[4] earth.google.com
[5] gdal.org
[6] www.r-project.org
[7] www.python.org

in each step [Sze10; GDA17]. For example, GDAL uses this to provide raster data access. If a raster layer with a resolution lower than the original layer is requested, GDAL looks for a matching pyramid layer. If the pyramid contains the requested resolution, the corresponding layer is returned instead. Tiling is another method for efficient raster access. The size of raster data depends on the resolution and the size of the covered area. With a size of $1,296,000 \times 648,000$ pixels the digital elevation data provided by the Shuttle Radar Topography Mission (SRTM)[8] very large. It provides elevation information for most of the earth's surface with a pixel size of 30 m×30 m. However, the elevation of pixels covered by large water areas is set to a no-data value. To avoid storing the large "empty" areas, the data is split into tiles of 3601×3601 pixels [Far+07]. This allows omitting the all-empty-tiles. Additionally, tiles allow for more efficient access to data storage. Similar to the approach of Quad- or R-Trees [Sam90], pixels in tiles are located near each other. This reduces required I/O operations if a subset of the dataset is requested. A sub-setting operation can be performed in two steps: First, only the relevant tiles are loaded, and second, required subset operations are applied only for the relevant tiles.

2.2.2 Distribution

Distribution of spatio-temporal data is mostly based on web protocols defined by the Open Geospatial Consortium (OGC). The Web Feature Service (WFS) allows accessing vector data and provides data set selection and filtering methods [Ope10b]. The Web Coverage Service (WCS) covers access to raster data [Ope08]. A particular profile called EO-WCS, where EO stands for Earth Observation, extends WCS for remote sensing data [BM11]. EO-WCS allows to request raster data for a user-defined spatial and temporal subset. The Web Map Service (WMS) provides raster and vector data as a rendered RGB image [Ope06]. All protocols have a multitude of parameters. Usually, a request specifies the identifier of the requested dataset and a spatial bounding rectangle. Temporal selection or restriction is only standardized for WCS.

2.2.3 Coordinate Reference Systems

When working with spatio-temporal data, handling of different coordinate reference systems (CRS) are often required. A CRS defines a frame to reference

[8] `www2.jpl.nasa.gov/srtm/`

locations of objects in space. There are two kinds of CRS, geographic and projected coordinate reference systems [De 13].

Geographic CRS use a three-dimensional representation of the earth. The representation is usually based on a spheroid, which abstracts the shape of the earth. It uses polar coordinates to indicate locations. The Longitude (Lon) represents the location along the equator, and Latitude (Lat) represents the angle between the equator and the poles. A location in Lat/Lon without height references a point at the earth's surface [De 13]. In the Simple Feature Model, for instance, the height is optional.

Projected CRS use a two-dimensional representation of the Earth in a Cartesian coordinate system. Polar coordinates are transformed into two-dimensional plane coordinates of the projected CRS. This uses the spheroid of the geographic CRS and a transformation rule. There are many kinds of projections, e.g., conical, azimuthal, or cylindrical. Transformations from projected into geographic coordinates is are also possible [De 13]. Map preparation and visualization often use Projected CRS; the default projection for online maps is a two-dimensional Mercator projection. Projected CRS are also significant for remote sensing data. Raster data produced by sensors on satellites is, caused by the nature of the raster model, represented in a two-dimensional coordinate system.

2.2.4 Visualization and Analysis

Visual globes like Google Maps[9] visualize spatial data as a three-dimensional globe, while desktop GIS like ArcGIS, GRASS, or QGIS usually present a projected two-dimensional map. This is also true for libraries like OpenLayers[10] or Leaflet[11] which can visualize spatial data on a map in a web-browser. To display large amounts of data, e.g., animal observations, data aggregation and visual clustering is required to present understandable visualizations [Bei+19b]. For raster data, selecting the matching aggregation algorithm and colorization is a key component for visualization on multiple zoom levels. Interactive maps, have to provide results fast to create a good user experience [AAG03].

GIS often provide toolboxes for data processing, e.g., operations for vector and image analysis in GRASS [NM13] and spatial statistics in ArcGIS [SJ10]. When data processing requires many steps, users can create workflows, where operators are chained, e.g., in QGIS [GO15]. Data scientists often require scripting

[9] `maps.google.com`
[10] `openlayers.org`
[11] `leafletjs.com`

languages to develop new methods, which is cumbersome using predefined tools. Therefore, languages like R or Python are used for vector and raster processing and analysis. The GDAL community provides libraries and interfaces to access spatio-temporal data in most languages. In Python, relational data, e.g., vector data, is often analyzed with Pandas[12]. For vector data, GDAL's OGR allows relational queries and other libraries, e.g., GeoPandas[13], provide additional operations. Raster data processing uses the powerful NumPy [vdWCV11] library, which enables n-dimensional array handling and functions like matrix multiplication. Using this toolset, data scientists can explore and analyze different spatio-temporal datasets.

2.2.5 Raster Operations

Operations for raster data are not defined by a single standard, which the Simple Feature Model does for vector operations. Domains like computer vision [Sze10], image processing, and remote sensing [RR99; Har06] use different sets of operations, which overlap with an image algebra defined by Ritter *et al.* [RWD90].

Pixels in a raster are defined as tuples (x, v), where $x \in X$ and $v \in V$. X is a set of coordinates, and V is a set of values. A grid A is composed from all pixels, where the pixel coordinates are mapped to pixel values $a = \{(x, a(x)) : x \in X\}$. The coordinates set X is defined as a subset of an n-dimensional Euclidean space. Since cells are addressed by rows and columns, $X \subset \mathbb{Z}^n \subseteq \mathbb{R}^n$ holds. Each coordinate x in a n-dimensional grid has n coordinate components $x = (x_1, x_2, \ldots, x_n)$ [RWD90].

Figure 2.2, shows examples for the three basic types of operations on raster data: Induced, set, and neighborhood operations [RWD90]. They are also found in an array algebra for database systems with focus on spatio-temporal raster data by Baumann [Bau99].

Induced operations are all operations that are valid on the datatypes of the raster cells and the cell coordinates. This includes operation on single pixels as well as operations between grids. An example is the addition of two grids which is defined as $a + b = \{(x, c(x)) : c(x) = a(x) + b(x), x \in X\}$.

[12] `pandas.pydata.org`
[13] `geopandas.org`

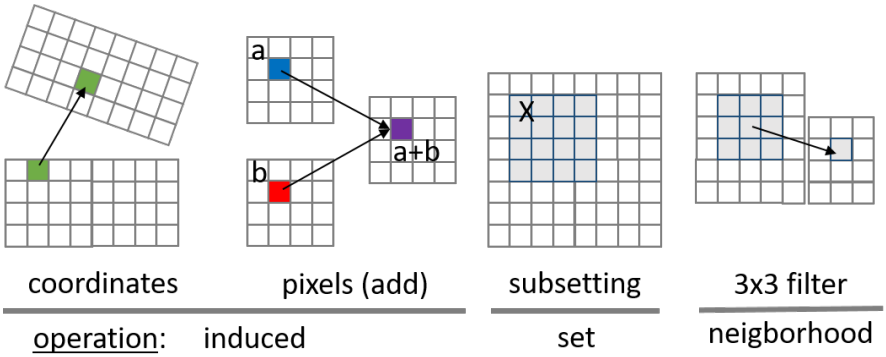


Figure 2.2: Examples for Raster operations. (a) shows induced operations, which include coordinate transformation and combination of two rasters. (b) shows the restriction of raster pixels as a set operation and (c) shows a filter, which is applied on a 3×3-pixel neighborhood.

Set operations are defined on raster values and the set of cell coordinates [RWD90; Har06; Bau99]. They include operations for unification, intersection, and restriction. The restriction is particularly important for raster data processing. It allows to restrict the coordinate set and therefore, the area of a raster to a subset, as shown in Figure 2.2. The restriction can also be applied to the value set of an image. This operation can be used, for example, as a filter operation that masks pixels using a threshold.

Neighborhood relationships include all operations applied over arbitrary neighborhoods of pixels. Ritter *et al.* [RWD90] present a powerful template operator, that combines and generalizes filters, aggregates and mask operations. Raster databases like Rasdaman [Bau99] use this operator type but split it to a set of several less complex operators. A simple example is to calculate the sum of all pixels of 3×3-pixel neighborhood, which is indicated in Figure 2.2. The filter is moved over all pixels, and the result is representative for the central pixel. Cell values of the filter can also represent weights, which enables operations like edge detection in computer vision [Sze10]. This is also relevant for Convolutional Neural Networks. An example for a specific filter is presented in Chapter 2.5.2.

Composition enables chaining of operations. Operator chains often use this property. Therefore, the composition of operations is of particular importance. While chaining of operations can materialize the results as a grid after each step,

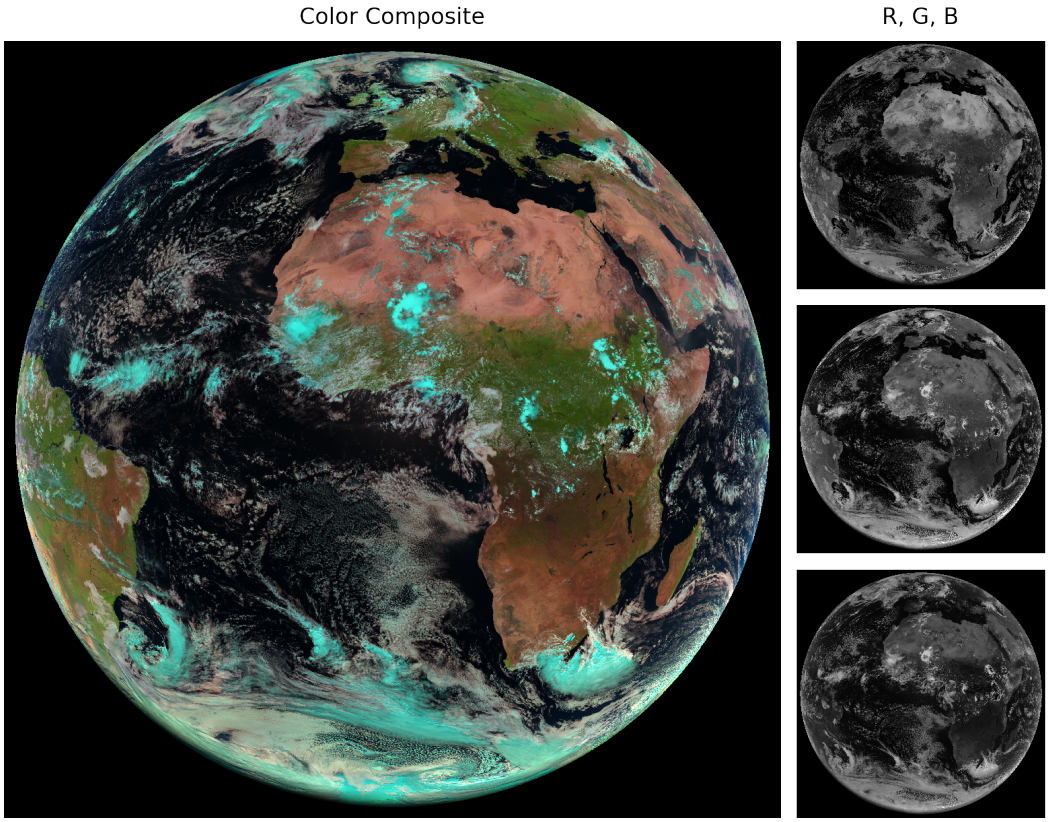


Figure 2.3: A color composite of the Red, Green, and Blue channels of MSG.

raster processing can also apply the chain of operators to pixels. This way, only the final grid is materialized [Har06].

2.3 Remote Sensing

Remote sensing by definition is the obtaining of information without being there [RR99]. Therefore, looking through a telescope is remote sensing. This is what remote sensing satellites actually do: The famous Hubble space telescope is an adaption of a military spying satellite. It inherited the large telescope used for military intelligence [CC98]. The most common approaches to remote sensing for earth observation use airplanes, drones, and of course satellites, which are the primary concern in this thesis.

The natural data model of remote sensing is the raster model, as introduced in Chapter 2.1. Satellite images showing RGB images of the earth, require three layers for red (R), green (G) and blue (B), to represent the reflected solar irradiation of the visible light. Figure 2.3 shows an RGB image generated from three channels of the European weather satellite Meteosat Second Generation (MSG). Next to the RGB composite, the three layers representing the R, G, and B signal received by the sensor on the sattlite are depicted. Besides the visible light, there is a broad range of interesting wavelengths in the electromagnetic spectrum ranging from microwaves to X-rays. A *spectral band* is a group of consecutive wavelengths, e.g., the visible light [RR99].

The resolution of satellite data has three components:

1. The **spatial resolution** is the size of a raster pixel on the surface, e.g. 100 m×100 m.
2. The **temporal resolution** indicates how often a new snapshot covering a location is created.
3. The **spectral resolution** depends on the width and the number of the observed spectral bands.

The spatial and temporal resolutions depend on the orbit of the satellite. While Low Earth Orbiter (LEO) allows a high spatial resolution of less than 100 m×100 m, GEOstationary (GEO) satellites provide a coarser pixel size, e.g., 1 km×1 km [RR99]. This is mainly caused by the distance to the earth surface, as shown in Figure 2.4.

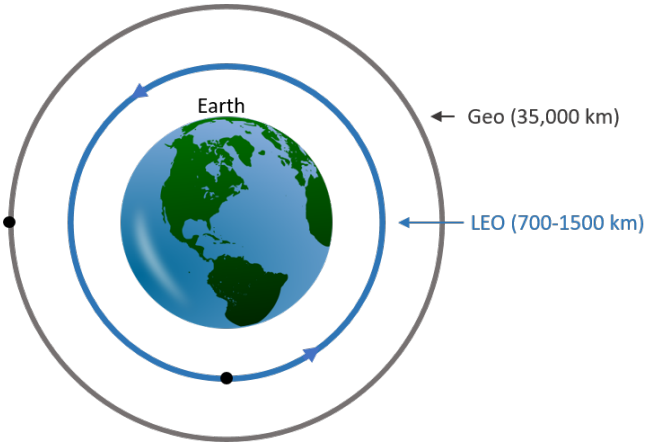


Figure 2.4: Satellites used for remote sensing are either Low Earth Orbiter (LEO) or placed in a Geostationary (GEO) orbit.

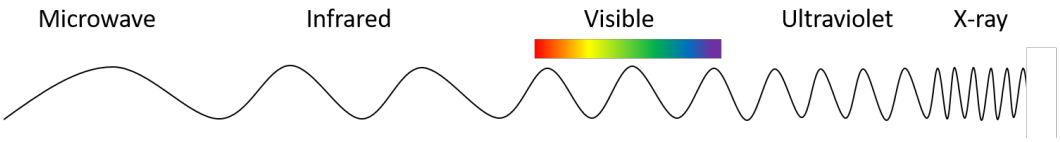


Figure 2.5: Visualization of the electromagnetic spectrum. The visual light, infrared and microwaves are used frequently for remote sensing.

LEOs are mainly placed at a height between 700 km and 1500 km. The often use polar sun-synchronous orbits, which causes the satellite to cross the equator always the same local time. However, this does not cause the satellite to capture the same area on each crossing. A satellite's revisit time measures the time between two overpasses allowing to capture the same area [RR99; Alb09].

The Moderate Resolution Imaging Spectroradiometer (MODIS) on the Terra and Aqua satellites is placed at the height of 705 km and provides a spatial resolution of 1 km×1 km. However, the sun-synchronous orbit only allows covering each pixel with a temporal resolution of 1–2 days [PWK06]. To improve this, MODIS is placed on two satellite platforms (Terra and Aqua), which allows covering a pixel twice each day. Similar to the MODIS approach, the SENTINEL-2 mission of the European Copernicus program uses two LEOs. They provide a spatial resolution of 10 m and a revisit time of 5 days at the equator under the same viewing conditions [Dru+12].

GEOs are placed at a geostationary orbit at approximately 36,000 km. MSG is placed at 35,786 km [Sch+02]. The geostationary orbit allows a satellite to always stay over the same location. Therefore, the temporal resolution depends on the scan speed of the sensor [RR99]. The European weather satellite MSG [Sch+02] or the Japanese Himawari [Bes+16] provide data with a very high temporal resolution. MSG at 0°N, 0°E, provides raster data covering a whole hemisphere (full disk), as shown in Figure 2.3. Africa, Europe, and the Atlantic Ocean are covered with a spatial resolution of 3 km resulting in a raster size of 3712×3712 pixels. The temporal resolution is 15 min. Himawari provides a fast scan mode for Japan. For this subarea of its full disk, a spatial resolution of 500 m and a temporal resolution of 2.5 min is available.

The sensors on the mentioned satellites cover more spectral bands than the visible light. In general, remote sensing data with more than one spectral measurement per pixel is defined as *multispectral*. Figure 2.5 shows a sketch of the distribution of wavelengths in the electromagnetic spectrum. The reflected solar radiation allows measuring signals from the visible and the near to middle infrared. Emission from the earth's surface in the thermal infrared range allows identifying properties

of observed surfaces also at night. The range of the wavelength in this band is between 0.4 µm and 12 µm [RR99]. An example of this is the multispectral main instrument of MSG satellites, which has 11 channels, covering bands in the visual and infrared spectrum [Sch+02].

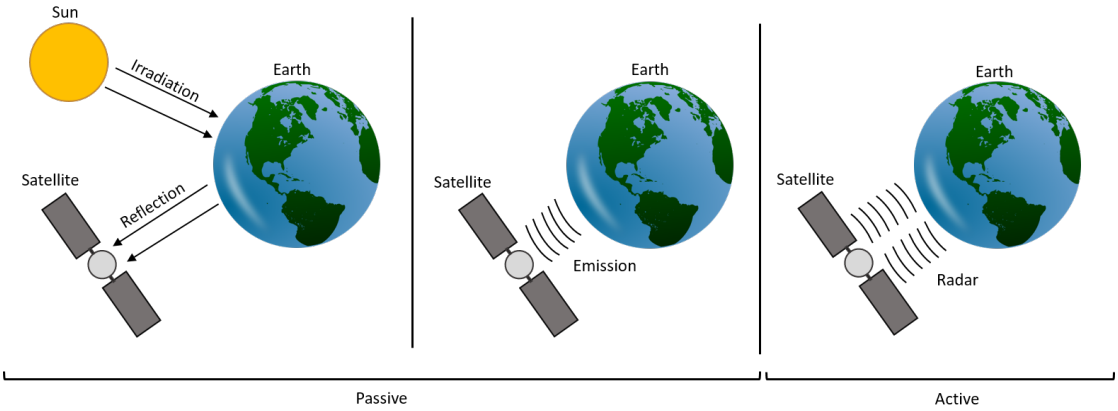


Figure 2.6: Visualization of active and passive remote sensing. On the left, the solar irradiation is reflected and received by a sensor. In the middle, a sensor, sensitive in the infrared spectrum receives emitted radiation of the earth's surface. The active approach on the right, sends signals, e.g., RADAR, and receives the echo.

Solar reflectance and thermal emission allow using *passive* sensors, which rely on an external radiation source. There are also active remote sensing systems, e.g., using RAdio Detection And Ranging (RADAR) with a wavelength between 30 mm and 300 mm [RR99]. Figure 2.6 depicts both kinds of sensors and the source of the measured radiation.

The importance of different wavelengths lies in the interaction between electromagnetic radiation and the observed material. In the visible and infrared range, the energy measured by a sensor depends on the reflection at the observed surface. The heat capacity and other thermal properties of the observed surface are relevant for the thermal infrared range. An active system uses, for instance, microwaves, to measure the roughness of the surface. Thus, each wavelength range has its purpose [RR99].

Active instruments using RADAR were used to create the popular SRTM digital elevation dataset [Far+07]. Two RADAR antennas on board of the Space Shuttle Endeavour were used in the year 2000 to acquire topographic data. Using the differences between the signals, a 1-arc-second global digital elevation model

(30 m) was generated [RR99]. As mentioned above, it has $1,296,000 \times 648,000$ pixels.

Remote Sensing of the Atmosphere is of particular interest for climate research and weather observation. An active remote sensing approach is used by CloudSat [Ste+08; PWK06]. CloudSat is designed to study the effects of clouds and aerosols on weather and air quality. Its primary instrument is the Cloud Profiling Radar, which uses the signal back-scattered by clouds to measure the distance of droplets from the radar. CloudSat is a LEO and flies in formation with other satellites, including Aqua, which carries the MODIS sensor. CloudSat also has a temporal resolution in the order of days [Ste+08; PWK06].

Remote sensing of the atmosphere for weather and climate relies on GEOs like MSG or Himawari. The temporal resolution of this kind of satellite provides time series with snapshots in the order of minutes, e.g., 15 min for MSG. Using the multispectral raster data, visual representations (RGB), indices, or classifications are produced. The next chapter (2.4.1) describes MSG, which is the main data source for this work. Additionally, applications are presented with a focus on cloud properties derived from MSG data.

2.4 Meteosat Raster Times Series

This chapter presents a short overview of the European Meteosat weather satellite program. Then, we discuss the raster time series and the properties of Meteosat Second Generation. Finally, a cloud product derived from the MSG data is presented.

The European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT) conducts the European weather satellite program Meteosat. The first Meteosat satellite was started in 1977 and provided a spatial resolution of 5 km $\times$ 5 km and a temporal resolution of 30 min. The MSG satellites took over in 2004. MSG's main instrument, the Spinning Enhanced Visible and Infrared Imager (SEVIRI), has a spatial resolution of 3 km$\times$3 km at the Sub-Satellite Point (SSP) at 0°N,0°E. The temporal resolution of MSG is 15 min and the spectral resolution was enhanced to 11 channels plus a panchromatic High Resolution Visible (HRV) channel [Sch+02]. In the future, Meteosat Third Generation (MTG) will increase all resolutions again [Rod+09]. However, the MSG time series still presents challenges and opportunities for remote sensing and raster time series processing.

2.4.1 Meteosat Second Generation

MSG SEVIRI provides multispectral scenes with a temporal resolution of 15 min [Sch+02], resulting in 96 scenes per day and 35,040 scenes per year. Because the data is available since the start of the MSG program in 2004, a large raster time series is available for tasks like cloud and fog detection. The operational satellite of the MSG program provides images of a hemisphere (full disk) covering the Atlantic, Europe, and Africa. Figure 2.3 presets an RGB image generated from the first three SEVIRI channels. The eleven main channels of SEVIRI are centered at wavelengths ranging from 0.6 µm to 13.2 µm while the additional panchromatic channel (HRV) covers multiple wavelengths in the visual and the near-infrared spectrum. The HRV channel has a different spatial resolution of 1 km×1 km. The produced raster has a size of $11,136 \times 5,568$ pixels. In the north, it is always centered at Europe, while for Africa, the area with the most solar irradiation is included [Sch+02].

The channels are named as a combination of their spectrum or absorption band (VIS, NIR, WV, IR) and the central wavelength (0.6 µm − 13.4 µm), as shown in Table 2.1.

Table 2.1: MSG SEVIRI channels [Sch+02].

#	Channel *	Central λ	Type	Remarks
1	VIS0.6	0.6 µm	Reflectance	Visualized as Blue
2	VIS0.8	0.8 µm	Reflectance	Visualized as Green
3	NIR1.6	1.6 µm	Reflectance	Visualized as Red
4	IR3.9	3.9 µm	Both	
5	WV6.2	6.2 µm	Emission	Water vapor absorption
6	WV7.3	7.3 µm	Emission	Water vapor absorption
7	IR8.7	8.7 µm	Emission	
8	IR9.7	9.7 µm	Emission	Ozone absorption
9	IR10.8	10.8 µm	Emission	
10	IR12.0	12.0 µm	Emission	
11	IR13.4	13.4 µm	Emission	Carbon dioxide
12	HRV		Reflectance	Broadband (0.4 µm–1.1 µm)

* official channel/band names for SEVIRI given by EUMETSAT.

Table 2.1 also shows additional aspects of the MSG SEVIRI channels. The visible (VIS) and the near-infrared (NIR) channels measure reflected irradiation and are dark at night, while Channel 4 covers both solar reflection and thermal emission during the day and only thermal emission during the night. The IR and Water

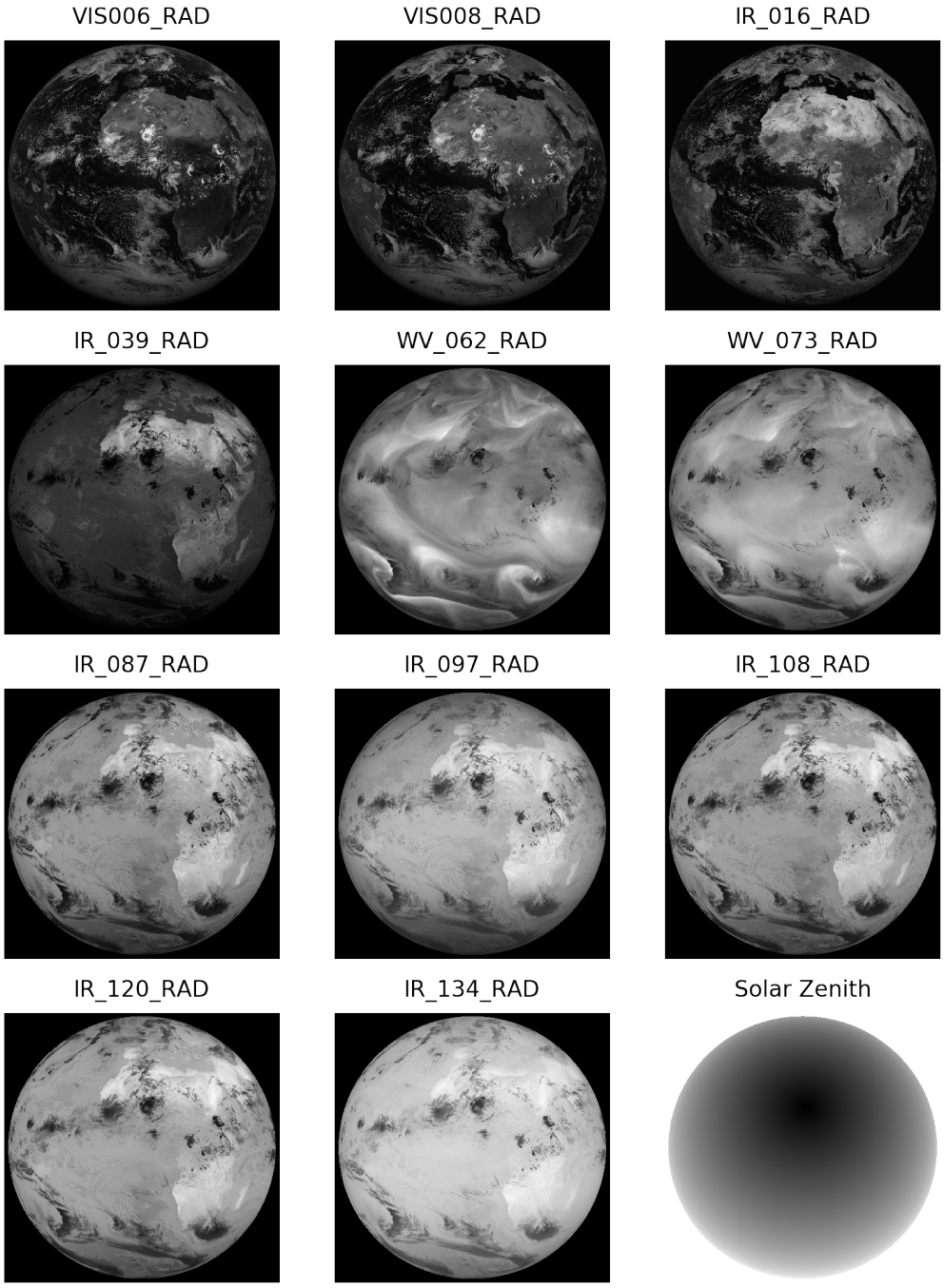


Figure 2.7: An overview of all SEVIRI channels on August 10, 2018 at 12:00:00 UTC. The channel names contain the sensitive spectrum or absorption band and the central wavelength in micrometers for each channel. Additionally, the bottom row on the right displays the SZA for the same date (black means small angle (0°), white a large angle (90°)).

Vapor (WV) channels measure surface emission in the mid and thermal infrared. At wavelengths of 6.2 µm and 7.3 µm, two channels cover the WV absorption band of the atmosphere, and the channel at 9.7 µm is sensitive to Ozone concentration [Sch+02]. Figure 2.7 presents an overview of the 11 main SEVIRI channels on August 10, 2018 at 12:00:00 UTC.

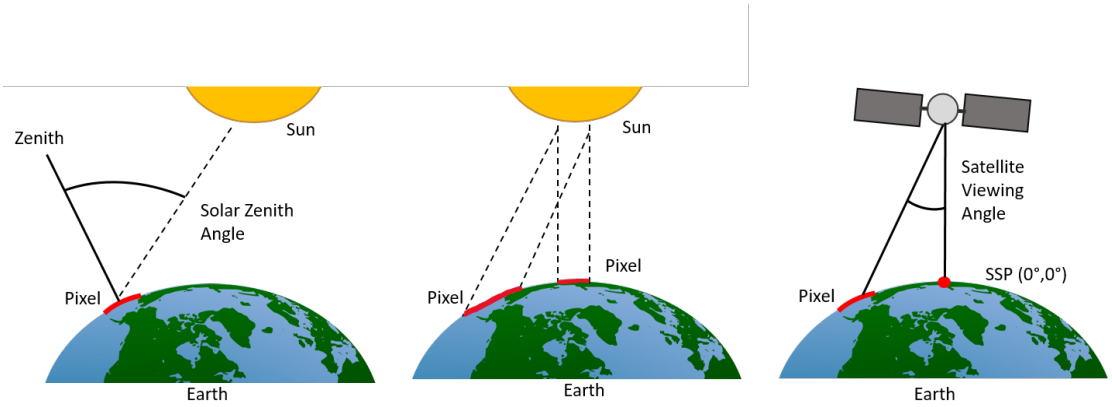


Figure 2.8: The different angles influencing the signal received by MSG SEVIRI. On the left, the SZA is depicted and the SVA is depicted the middle and on the right.

The Solar Zenith Angle (SZA) measures the angle between the zenith at each pixel and the virtual line connecting the pixel and the sun. It is relevant for the amount of solar irradiation, as indicated by Figure 2.8. A SZA of 0° indicates that the sun is directly above that pixel and a maximum of irradiation arrives at the surface. With a larger angle, the irradiation is distributed to a larger area. The SZA is also displayed by Figure 2.7 at the bottom row on the right (black means small angle (0°), white a large angle (90°)).

The second angle is the Satellite View Angle (SVA), as presented in Figure 2.8 This is the angle between the virtual lines connecting MSG with the SSP and MSG with each pixel. The geostationary position of the MSG satellites causes this angle to stay fixed for each pixel. It depends on the location of a pixel and increases with its distance to the SSP.

The SZA and the SVA cause two essential effects [Cer06]. Increasing the SVA also increases the distance between pixels and MSG, which distorts the actual pixel size on the surface. Over Europe, pixel sizes grow up to 11 km. At the same time, the column of air, and thus the amount of absorbing gases such as CO_2, increases with a growing SVA. The SZA causes the second effect. While the geostationary position of MSG causes SVA to remain constant, the SZA changes with the diurnal

cycle of the sun. This affects the actual solar radiance and therefore, the reflection detected by SEVIRI during daylight. Channels 1–3 pose challenges caused by the SZA [Cer06]. These channels are dark at night and therefore, do not provide information. At daytime, sunlight might get reflected by water surfaces causing glare (sun-glint) effects [HFS10]. Channel 4 changes its characteristics depending on solar irradiation. While the IR signal is dominant at night, the reflection of solar irradiation is dominant at daytime. To handle these effects, classification approaches often require rasters with the SZA (Figure 2.7) and the SVA as an input [Cer06].

MSG SEVIRI data is available from EUMETSAT, via multiple services. It's collection reference is *EO:EUM:DAT:MSG:HRSEVIRI*[14]. The High Rate Information Transmission (HRIT) format stores both, raw data and metadata. The raw data generated for one SEVIRI scan is about 2.4 Gbits (0.3 GB) [Sch+02]. Since the MSG program started in 2004, the time series grows by approximately 8 GB of compressed data per day. The available data is geolocated and it is corrected for most radiometric and geometric effects [EUM10b]. The HRIT format was specially developed for geostationary weather satellites and also determines the projection of the data. Since the raw data does not correspond to a physical unit, all channels require transformation using multiple preprocessing steps, which are discussed in Chapter 4.2.

2.4.2 CLAAS-2 Cloud Mask

The CLoud property dAtAset using SEVIRI - Edition 2 (CLAAS-2) dataset [Ben+17; CM 16] provides different cloud properties derived from MSG SEVIRI data, including a Cloud Mask (CMa), cloud top height, and cloud top temperature for the 12 years from 2004 to 2015. Figure 2.9 shows the CMa next to the RGB composite for an area centered on Europe.

The CMa provides a pixel-wise classification into four classes [DGF13]:

1. Cloud-free: The pixel is not contaminated or filled by clouds.
2. Cloud-contaminated: The pixel is partly cloudy (mixed) or filled by semi-transparent clouds.
3. Cloud-filled: The pixel is completely filled by opaque clouds.
4. Snow/ice contaminated: The pixel contains snow or ice.

[14] `navigator.eumetsat.int/product/EO:EUM:DAT:MSG:HRSEVIRI`

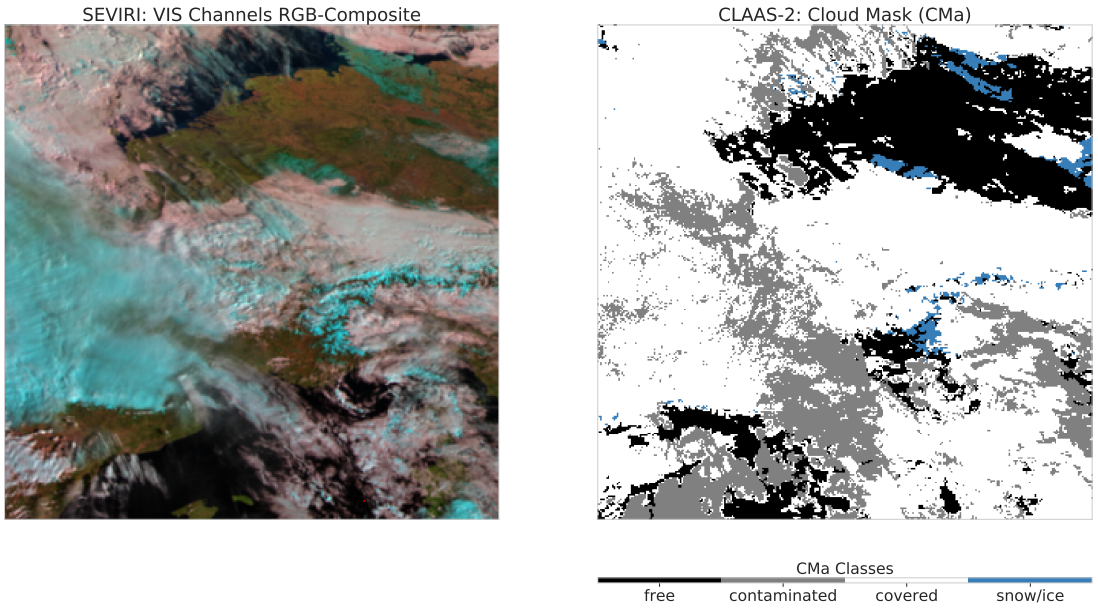


Figure 2.9: Visualization of the RGB composit and the CLAAS-2 Cloud Mask (CMa) on February, 22 2011 at 09:00:00 UTC. The image on the left shows the RGB composite centered on Europe, including North Sea and and the Alps covered by snow. The CMa, covering the same area, is displayed on the right.

The algorithm applied to generate the CMa data is a modified version of the SAFNWC-MSGv2012 algorithm [DGF13]. The Satellite Application Facility on Climate Monitoring (CM-SAF) applies this algorithm for nowcasting. A short runtime is required to deliver the products generated by classification or other processing steps as quickly as possible. Therefore, the CMA algorithm is run on aggregated pixels for nowcasting applications to lower the required runtimes. For the generation of the CLAAS-2 data set, however, the algorithm was applied for each pixel individually [DGF13; Ben+17].

The CMa algorithm includes several threshold tests as well as spatial tests. For spatial tests, a pixel's neighborhood is aggregated, e.g., 3×3 pixels. The result of the selected aggregation method is then used to classify the pixel. Pixels are first classified as cloud-contaminated using threshold values. If the value of a pixel is sufficiently distant from the threshold, it is classified as cloud-filled [DGF13].

Tests distinguish between conditions of changing solar irradiation (day and night), which are caused by the SZA. Additionally, separation is done for land and sea as both have different spectral properties [DGF13]. This allows using specific

thresholds for effects such as sun-glint over the sea or the different properties of the 3.9 µm channel at day and night (see Chapter 2.4.1). Different thresholds are used to solve the challenges posed by various effects, which includes the SZA and SVA.

Validation studies are available for the CLAAS-2 dataset [Ben+17]. The validation study used data from active and passive instruments on low earth orbiters. The Cloud-Aerosol Lidar with Orthogonal Polarization (CALIOP) is an active instrument, while MODIS is a passive instrument. The study shows that the CLAAS-2 data, including the cloud mask, agree overall very well with the reference datasets and that no severe discrepancies were found. However, since the technologies of the reference platforms and SEVIRI differ significantly, no perfect pixel matching is possible.

Some known technical limitations include classification of snow and ice contaminated pixels. The corresponding rules in the CMa algorithm require reflectance information from the solar Channels 1–3 to classify snow. Thus, the snow/ice class is only available for pixels with solar irradiation and therefore only at daytime [DGF13]. The CMa algorithm incorporates the Normalized Difference Snow Index (NDSI) [Doz89], which is defined as

$$\text{NDSI} = \frac{\text{VIS0.6} - \text{NIR1.6}}{\text{VIS0.6} + \text{NIR1.6}} \tag{2.1}$$

to identify snow pixels. The actuall snow detection involves at least seven combined tests and the NDSI is calculated for each pixel. If the NDSI is above $0.3 + 0.15$, where 0.15 is normalized using the SZA, a pixel is classified as snow. Snow pixels are indicated by a blue color in the CMa image shown in Figure 2.9.

2.4.3 Open Computing Language (OpenCL)

Graphics Processing Units (GPU) have undergone a significant evolution from a static *graphics pipeline* implemented in hardware to freely programmable parallel processors. This also causes the increasing application of GPUs for classification of raster and image data. The use of GPUs to process arbitrary data is called General Purpose GPU (GPGPU) computing [Owe+08]. There are mainly two programming languages used for GPGPU computing. While CUDA is a proprietary language by Nvidia[15], the Open Computing Language (OpenCL) is an open

[15] `nvidia.com`

standard, endorsed by Intel and AMD. Both are based on the Single Instruction Multiple Data (SIMD) model and also have high similarities in their syntax [Gas+11].

OpenCL provides a heterogeneous computing model that permits applications to run on SIMD hardware, e.g., multi-core CPUs with large SIMD vector units, and on SIMD-like hardware, such as the Single Instruction Multiple Thread (SIMT) architecture of GPUs. On CPUs, each core represents a independent Compute Unit (CU), which allows all CUs to execute independent instructions. A CU on a GPU controls a group of threads executing the same instructions. Work in OpenCL is represented as a grid of work-items, which are partitioned into sub-grids called work-groups. The program executed by CUs is called *Kernel*. When a Kernel is executed, the work-groups are spread over the available CUs that process the contained work-items. Pixel-wise raster processing with OpenCL is equivalent to splitting a raster of pixels into tiles, each representing a work-group. Then, each tile is mapped to a CU where the pixels (work-items) are processed. In general, a thread is executed for each work-item in the global work-size, and a local work-size determines how many threads each CU runs concurrently [Khr12; Gas+11].

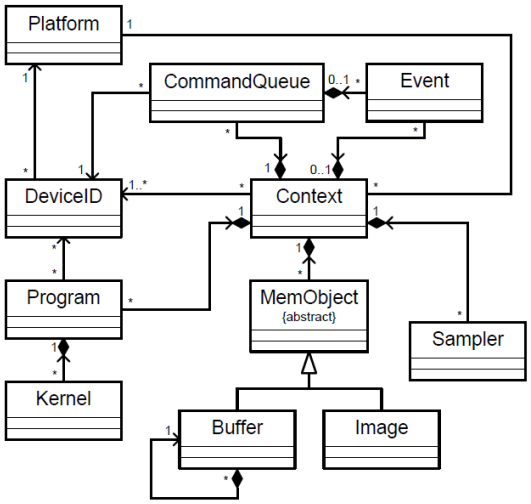


Figure 2.10: UML-Diagram of OpenCL [Khr12]

Figure 2.10, depicts the structure diagram of OpenCL. Since it is an open standard, different implementations are available. OpenCL-*Platforms* can support multiple device types, e.g. GPUs and CPUs. In practice, several platforms can exist in one system. To use OpenCL, a *Context* is created for a selected platform, which in turn contains a number of devices [Khr12; Gas+11].

Programs are written in a C-based language and are executed in a selected context. The OpenCL specification allows that Programs are translated into executable Kernels at runtime [Khr12; Gas+11].

Due to the execution paradigm of OpenCL, if a Kernel is to be executed, it is first inserted into a *CommandQueue*. The Kernels and other operations in a CommandQueue are processed in the insertion order. However, it is also possible to make a Kernel explicitly wait for a group of *Events*. The execution or termination of a Kernel, but also the transfer of data to a device are events [Khr12; Gas+11].

Creating a Context and inserting a Kernel into a CommandQueue is the task of the host system, which manages the execution of Kernels [Khr12; Gas+11]. Libraries for OpenCL are available for many programming languages, e.g., Rust[16] or Python[17]. Code Listing 2.1 shows how an OpenCL Context is created, and a Kernel is enqueued with Python. The example uses the PyOpenCL library [Klö+12] and the Numpy library [vdWCV11] for array data handling.

Listing 2.1: Python and OpenCL Example

```
1 import numpy as np
2 import pyopencl as cl
3
4 # generate random data
5 arr_1 = np.random.rand(10000).astype(np.float32)
6 arr_2 = np.random.rand(10000).astype(np.float32)
7
8 # create an empty array for the result
9 arr_res = np.empty_like(arr_1)
10
11 # create Context and CommandQueue
12 ctx = cl.create_some_context()
13 queue = cl.CommandQueue(ctx)
14
15 # create OpenCL Buffers and copy data to the Device
16 mf = cl.mem_flags
17 gpu_arr_1 = cl.Buffer(ctx, mf.READ_ONLY, hostbuf=arr_1)
18 gpu_arr_2 = cl.Buffer(ctx, mf.READ_ONLY, hostbuf=arr_2)
19
20 # create a Buffer for the result
21 gpu_arr_res = cl.Buffer(ctx, mf.WRITE_ONLY, arr_res.nbytes)
22
23 # Create a Kernel for the Context
24 prg = cl.Program(ctx, """
25 kernel void sub(
```

[16] crates.io/crates/ocl
[17] mathema.tician.de/software/pyopencl/

```
26     float *gpu_arr_1, float *gpu_arr_2, float *res)
27 {
28   int gid = get_global_id(0);
29   res_g[gid] = gpu_arr_1[gid] - gpu_arr_2[gid];
30 }
31 """).build()
32
33
34 prg.sub(queue, gpu_arr_1.shape, None, gpu_arr_1, gpu_arr_2,
      gpu_arr_res)
35
36 cl.enqueue_copy(queue, arr_res, gpu_arr_res)
```

In Line 1 and 2 the Numpy and PyOpenCl libraries are imported and aliased. We generate random data for two arrays with a size of 10,000 elements in Lines 5 and 6. An empty result array is initialized in Line 9 with the same size as the other two. In Line 12 an OpenCL Context is created. A CommandQueue for this Context is created in Line 13. The array data of the arrays *arr_1* and *arr_2* are mapped or copied (this depends on the Device) into OpenCL Buffers in Line 17 and Line 18. A new empty Buffer with the size of *arr_res* is created on Line 21.

The Lines 24 to 31 define a simple OpenCL Program with a single Kernel "sub". We define three parameters as input for the Kernel. They are all float arrays. In Line 29 the built-in OpenCL method *get_global_id* is used to determine the id of the thread in the global work-size. Using the id as the position in the arrays in Line 29, the float values from the second input array are subtracted from the first. We save the result in the array provided as a third input parameter.

Line 31 triggers building the OpenCL code, and in Line 34 the Kernel "sub" is enqueued into the CommandQueue. The Kernel has multiple parameters. First, the CommandQueue is provided where the Kernel is enqueued. Second, the global work-size is provided, which determines how many threads will be executed. In this case, this is the size of the arrays. Therefore, we subtract all elements. The fourth parameter is not used in this example, but it can provide a local work-size, which we explain later. Then, the two inputs and the empty buffer for the results are provided. In Line 36, the results are copied into the empty array. The CommandQueue performs this operation after the execution of the Kernel finishes.

Raster Processing Pixel-wise raster processing with OpenCL is equivalent to splitting a two-dimensional raster into tiles, each representing a work-group. Then, we map each tile to a CU where the pixels (work-items) are processed. In the following chapters, the OpenCL language is used for pixel-wise raster processing as well as complex operations such as histogram generation. The use-cases are explained in the corresponding chapters.

2.5 Machine Learning

This section presents two classification algorithms from machine learning, which are relevant for this thesis. The classification of raster data assigns a class from a predefined set of classes to each pixel. While there are many different algorithms applied for raster time series processing, Random Forest (RF) [Bre01] and Convolutional Neural Networks (CNN) [LBH15] are essential for the following chapters. Using decision trees to classify raster data is an obvious choice, which correlates with the rules used for CMa generation. Rule sets developed by humans often follow a tree structure. CNNs are the second method used in this thesis. They are built to classify and segment data represented as n-dimensional arrays which map perfectly to raster data. The other benefit is that their design allows using GPU processing, and therefore processing a very high amount of data.

2.5.1 Decision Trees

An example for a hand-crafted decision tree is the cloud classification used to create the CLAAS-2 CMa presented in Chapter 2.4.2.It includes many rules, two of them are the following: If the surface observed by a pixel is warm, it is probably not a cloud, and the pixel is rejected. In the next step, cold pixels are checked for snow. If the NDSI indicates snow, the pixels are also rejected.

The example uses a rule set developed by domain experts. Algorithms also allow generating rules by training decision trees from example data. Decision trees have many benefits. The computational cost of using a tree for classification is O(log(N)), where N is the number of data-points if the tree is balanced. The worst case is O(N) if the tree degenerates. Additionally, they are easy to understand because the path of a classification resembles human behavior [Mar14].

The construction of a decision tree from data is done in a greedy manner, starting at the root of the tree. The feature, which provides the highest information gain,

is selected at each split. This requires a measurement of how informative a feature is. One approach is to use the entropy, which describes the impurity of a set of features [Mar14].

The total entropy of a set of probabilities p_i is defined as

$$\text{Entropy}(p) = -\sum_i p_i \log_2(p_i),$$

where we assume a binary split. Here, p_i is the probability of data-points that belong to a feature i at a split. The entropy is therefore 0 if all data-points are classified identically. A split with $p_i = 0.5$ results in the highest gain of information [Mar14].

Another approach uses the *Gini Impurity* to select the features. The goal here is that a leaf node in a tree should contain as many data points as possible that are identical concerning a feature. If a leaf is pure, then all contained training data have only one feature. To calculate the Gini Impurity, we first assume that the data-points are randomly labeled according to the feature distribution. Then, the Gini Impurity is the probability of incorrectly classifying a randomly chosen data-point. Therefore, we sum up the probability of a data-point p_i belonging to feature i, times the probability of a false categorization. Concerning a set of features k we get the Gini Impurity

$$G_k = \sum_{i=1}^{k} p_i(1 - p_i) = 1 - \sum_{i=1}^{k} p_i^2.$$

By maximizing the difference between each value G_k and the total Gini Impurity, we chose the features for the best split [Mar14].

Random Forest

Random Forest (RF), as introduced by Breiman [Bre01], is an ensemble learning method combining a large number of standard decision trees. To combine the trees, a technique called *bagging* is used, which stands for bootstrap aggregation. A bootstrap sample is a sample taken from the original data with replacement. This implies that some data-points are selected multiple times, while others are never used. In an RF-model, each tree is trained separately by taking a bootstrap sample from the training dataset. The RF adds additional randomness besides the bootstrap approach. When training, subsets of features with a fixed size are selected from the original features. The trees are then built using the random

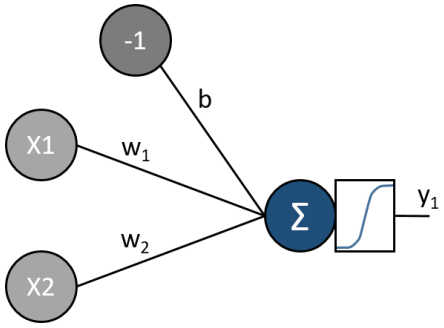


Figure 2.11: A visualization of a neuron. A neuron has multiple inputs ($x_1 \dots x_m$) with weights ($w_1 \dots w_m$). The additional input is set to a constant value with a weight called bias (b). The sum of the weighted inputs is calculated by the neuron and an activation function is applied, which is visualized as box. The output of the neuron y represents the result.

subset of features to split each node. The split is executed in such a way that within each subset of features and data-points, a chosen error function is minimized. Usually, RF uses the Gini Impurity to select the feature to use at each split. This procedure is recursively repeated until the complete training dataset has been processed [Mar14].

RF is straightforward to use and can easily handle vast numbers of input features and supports large datasets due to its sampling approach. To classify input data, RF uses an aggregation method. The result of all trees are collected, and a majority vote is used to determine the result [Mar14].

2.5.2 Artificial Neural Networks

Artificial neuronal networks are a computational model inspired by the function of real neurons. First, we present the basic concepts of neurons. Then, feedforward neural networks are discussed with a focus on Convolutional Neural Networks (CNN).

Neurons

Figure 2.11 introduces the basic concept of a single neuron. It receives inputs from other neurons (x_1 and x_2). Each connection has a weight (w_1 and w_2), which are assigned based on the importance of the inputs. There is also a third input, with

a fixed value and a weight b, which is called *bias*.A neuron calculates the sum of m inputs multiplied by the weights

$$h = \sum_{i=0}^{m} w_i \cdot x_i + b.$$

In practice, the weight of the bias is trained like all other weights. Then, an activation function is applied to the sum, which recreates the "firing" of a real neuron, i.e., if it should send a signal to its output. In neural networks, activation functions always return a value. This is either a binary 0 or 1 signal, or a value range. The often-used *sigmoid* function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

takes a real-valued input and transforms it into a value in $]0, 1]$. For the CS-CNN in Chapter 7 we use the *rectified linear unit* (ReLU)

$$f(x) = \max(0, x),$$

that has the advantage to return a value (≥ 0) and to be piecewise linear [Mar14; SB14; GBC16].

Feedforward Neural Networks

Feedforward neural networks are artificial neural networks, where connections between neurons do not form cycles as displayed in Figure 2.12. It is also called perceptron. The left side shows the inputs of the net. These are not neurons but indicate how many inputs the net uses. Therefore, information flows from input nodes through optional hidden nodes to output nodes. The stack of input values is also called *input vector*, which can be modeled as an array. The input data determine the number of inputs, and the number of neurons is independent. The network learns by supervised training, which means that it reproduces a target, which is encoded as an array of values mapped to the output range [Mar14].

Single-Layer Perceptron A simple network may use only one layer, which directly provides its results as outputs. To enable this network to learn, the weights and the activation function are the parameters to tune. The neurons are independent of each other, and all inputs are connected to all neurons. The connection

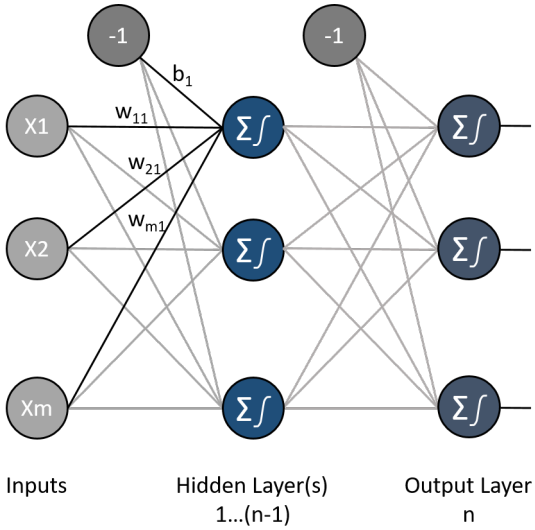


Figure 2.12: A feedforward neural network. Each neuron is connected to all inputs and uses individual weights to calculate the weighted input sum. Then, the activation function (visualized in each node) is applied. A variable number $n-1$ of hidden layers is placed between inputs and the output layer. If the output layer is the only layer there are no hidden layers.

from an input i to a neuron j has the weight w_{ij}. Using input data and a known result pattern, one can determine for each neuron what value it should produce. If a neuron k reacts wrong, the weights w_{ik} must be adapted. Therefore we calculate the difference between the output of the neuron y_k and the expected result t_k, which is an error function. To handle negative values and control how much the weight is changed, the learning rule is defined as

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - tj) \cdot x_i,$$

where η is the learning rate and x_i is the input value. The learning rate makes the network more resistant to noise as the weights are changed slower. However, this also requires the network to use an increased amount of training samples [Mar14]. The single-layer perceptron is only able to learn weights for linear models. For more complex tasks, e.g., the XOR-function, neural networks with more layers of neurons are required [Mar14].

Multi-Layer Perceptron A Multi-Layer Perceptron (MLP) has more than one layer of neurons. The layers between the inputs and the output neurons are called

hidden layer. Remember, that the input layer is just a visualization of the input data and does not contain neurons. Going forward through an MLP is the same as for the single-layer perceptron. With an input vector, the weights and activation function in the network are used layer by layer to calculate the result-pattern. However, training the network is much more difficult. First, we calculate the differences between input and output. One possible error function to measure the error of the network is the sum of squared errors, i.e.,

$$E(t, y) = \frac{1}{2} \sum_{k=1}^{n} (y_k - t_k)^2,$$

where t_k is the target, n is the number of neurons, and y_k is the neuron's output [Mar14].

Back-propagation of error is used to calculate the errors of neurons layer by layer from the outputs to the first hidden layer. This relies on differential functions and the chain-rule. Therefore, the activation functions of the neurons are required to be differential, e.g., the sigmoid function. Initially, the weights in the network are set to small random numbers. The key to weight the adaption is using a gradient descent method. Differentiating the error function concerning the weights gives the gradient of the error. The task is to change the weighs in a way that is towards the optimum of the gradient. Since it is impossible to directly differentiate the error function for the hidden layers, the chain-rule is applied to solve this [Mar14].

The weight adaption works like this [Roj13]: For a count of l weights in a network, the gradient of the error is

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_l}\right).$$

For each weight w_{ij}, an increment of

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

is applied to update its value

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}.$$

The MLP with multiple layers, which build on top of each other, is the base model for CNNs [GBC16], which are presented next.

Convolutional Neural Networks

Convolutional Neural Networks are specialized on data with a grid-like structure, e.g. raster data. The MLP with multiple layers, which build on top of each other, is the base model for deep learning and CNNs builds on-top of this [GBC16]. They are also used to process remote sensing data, which is represented as raster data similar or sometimes identical to images [BAC17]. In this section, we first look at the application of CNNs for image classification. The idea when using CNNs for this task is to go through a training process that contains images and the associated labels. For example, we assume a training set that contains thousands of pictures of dogs, cats, and birds. Each of the pictures has a label of the depicted animal.

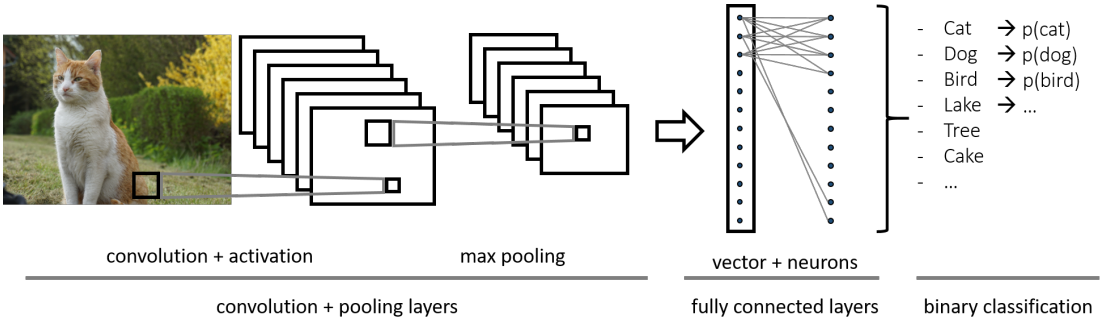


Figure 2.13: The picture of a cat and the basic concept of using CNNs for image classification. The convolution and activation layers are connected to fully connected layers, which provide a binary classification similar to MLPs.

Figure 2.13 presents the different stages of a CNN for image classification. It also shows the structure of neural networks as presented above, and the idea of weight adaption uses the same back-propagation strategy. While neural networks connect all neurons of a layer with all the neurons of the following layer, the CNN uses an idea inspired by the visual cortex. As shown in Figure 2.13, a neuron in a convolution layer only *looks* at a limited area of the input grid, e.g. the cats tail. Let's assume, that the input is an RGB image with 508×508 pixels, therefore the input consists of $508 \times 508 \times 3$ values. For this image, a number of filters (or kernel) are applied. If a kernel with the size of 7×7 cells is used, it also has to have the same depth as the input ($7 \times 7 \times 3$ cells). The window *seen* by the kernel (neuron) is also called the receptive field. Moving the kernel over all cells with a fixed step size is the convolution operation [GBC16].

A kernel function might be for edge detection or for the detection of a color. For the

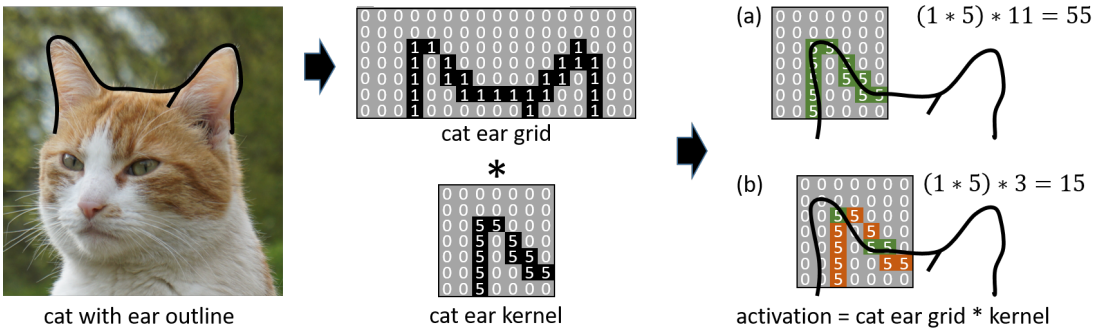


Figure 2.14: An example of a kernel for detecting the right ear of cats. The cat's silhouette is represented as a 1-dimensional grid where the outline has values of 1 while other cells are 0. The kernel has cells with weights of 5, which resemble the shape of right cat ears. Applying the kernel on the right location (a) results in a weighted sum of 55 while on another location (b) the sum is only 15.

cat in the example, an interesting kernel might be one detecting the characteristic shape of cats' ears, as visualized in Figure 2.14. In this example, we assume an input grid representing the cat's outline. It represents the outline with cell values of 1. The kernel cells represent the weights, similar to the connections of the neuron in the MLP. Using the weights, we multiply each cell in the receptive field with the overlaying kernel cell and add the results to each other. As shown in Figure 2.14, the resulting weighted sum is much larger if the structure is detected. This summation is equivalent to the summation used by the neuron in the MLP. The result of this operation is an activation map (which is also called feature map) with a size of $502 \times 502 \times 1$ pixels. The x and y dimension shrink since the kernel produces a single value for its center. The kernel size of 7 pixels means that 3 pixels are not represented in the result, except if the border pixels are padded. The z dimension shrinks since only one kernel is used, the z dimension represents the number of kernels, which is variable. The effect of the receptive field is that the neurons in a CNNs are sparsely connected, while a neural net usually connects all layers. A kernel also uses the same weights for all convolution operations, which is called shared weights. This reduces the size of the model. Chaining filters enable a CNN to learn higher hierarchy concepts, e.g., a cats head with two ears [GBC16].

In a CNN, the convolution operation and the activation function are often modeled as individual layers. An activation function, e.g., the sigmoid function, introduces non-linearity and is applied to the cells of the activation map, which is composed of the results from the convolution operation. This setup is similar to the neuron

in Figure 2.11, where the activation function is represented as a separate step. An optional third layer, after convolution and non-linearity, is called *pooling*, as indicated in Figure 2.13. A pooling layer also has a receptive field. However, the applied function is simply an aggregation, like max, min, or average. This is done to compile the representation invariant into small translations in the input [GBC16].

To classify an image, we add a fully convolutional layer to the last convolution, non-linearity, or pooling layer. It connects all the cells to a number of neurons, which is identical to the number of classes the net should detect. Similar, to MLP, the output of each neuron would represent the probability of each class, e.g., "cat" for the example in Figure 2.13.

Training a CNN uses back-propagation in the same way as the MPL does. First, we calculate the error of the forward pass, e.g., using the squared error sum. Then the weights are adapted layer by layer [GBC16].

Figure 2.15: Examples for object detection and image segmentation. On the left, the bounding box of a cat is indicated by a red square. On the right, the pixels in the image are classified as cat or non-cat.

Detection and Segmentation Besides image classification, computer vision has additional objectives. Two of them are object detection and semantic segmentation. The images in Figure 2.15 show an example for the already presented cat from Figure 2.13. In object detection, bounding boxes which enclose the area of the object are produced, i.e., the square area containing cats. In segmentation, the image is pixel-wise classified [Sze10]. The example in Figure 2.15 shows this for the cat. CNNs work very well for both tasks, e.g., the U-Net [RPB15] for segmentation, and the R-CNN [He+17] for both tasks.

Evaluation Metrics

We use the following concepts to test the cloud classification models, which are trained using RFs and CNNs in this thesis. First, we generate a contingency table by comparing results with the targets pixel by pixel. The idea of the confusion matrix is to use a quadratic matrix containing all possible classes in both horizontal and vertical directions. Along the top of the matrix, all actual classes are listed and the left side lists the predicted classes. At the position (i, j) in the matrix, one can see which elements belonging to class i have been sorted into class j by the model. Everything sorted along the diagonal of the matrix, i.e., where $i = j$ is valid, is correct [Mar14].

		Actual Class	
		True	False
Predicted Class	True	True Positive (a)	False Positive (b)
	False	False Negative (c)	True Negative (d)

Figure 2.16: The confusion matrix

To get more information by class, we calculate the confusion matrix. The confusion matrix, as depicted by Figure 2.16, contains four counters. True positives (a) are the correctly predicted events, i.e., the correct classified pixels. The incorrectly predicted events are the false positives (b). False negatives (c), are the incorrectly predicted no-events. Finally, the true negatives (d) are the correctly predicted non-events.

Using this information, we can calculate multiple metrics. The metrics are described in detail by Marsland [Mar14]. The accuracy of a class is the percentage of correctly classified positive and negative pixels

$$\text{Accuracy} = \frac{a + d}{a + b + c + d}.$$

The Probability of Detection (POD) or recall returns the percentage of pixels that correctly belong to a class

$$\text{POD} = \frac{a}{a + c}.$$

The Probability Of False Detection (POFD) gives the relative amount of pixels falsely classified

$$\text{POFD} = \frac{b}{b+d}.$$

The False Alarm Ratio (FAR) provides the probability of a false classification of a pixel classified as belonging to a class

$$\text{FAR} = \frac{a}{a+b}.$$

Additional metrics for forcasting are found in Jolliffe and Stephenson [JS03]. This also includes the Heidke Skill Score (HSS), which reflects discrimination and reliability to measure the improvement in comparison to a (random) standard classification. EUMETSAT's training project EUMETRAIN[18] provides a simplified formula. It defines the HSS as

$$\text{HSS} = \frac{2(a \cdot d - b \cdot c)}{(a+c) \cdot (c+d) + (a+b) \cdot (b+d)}.$$

Before presenting our models for the classification of clouds and fog from raster data, the next chapter presents related work. Besides the classification of satellite data, we look at raster processing and visualization.

[18] http://www.eumetrain.org/data/4/451/english/msg/ver_categ_forec/uos3/uos3_ko1.htm

3
Related Work

This chapter presents related work for the different aspects of this thesis. First, we cover related work in efficient and parallel raster processing in Section 3.1. Then, Section 3.2 covers cloud and fog detection in the remote sensing domain. Third, Section 3.3 presents machine learning techniques used to classify remote sensing data. Last, Section 3.4 focuses on the visualization and analysis of spatio-temporal data for explorative workflows.

3.1 Raster Processing

There are many applications for raster processing, e.g., for microscopic images [SRE12] or remote sensing [RR99]. For raster processing two aspects are important: efficient access and fast processing of the pixels.

3.1.1 Raster Data Management

The aspects of data access are mostly addressed by array database systems. Rusu and Cheng [RC13] present a survey on array storage, query languages, and systems, including Rasdaman [Bau+98] and SciDB [Sto+11]. All considered systems partition arrays into chunks for storage across a single or multiple disks. Indexing and on-demand re-partitioning allows additional speedups.

The raster data manager Rasdaman is dedicated to n-dimensional array data, which is called Multi-dimensional Discrete Data (MDD). Examples for MDDs are two-dimensional satellite raster, four-dimensional temporal data cubes from climate models, or one-dimensional measurements. The declarative query language *rasql* enables operations on MDDs; they adhere to the algebra referenced in Chapter 2.2. Rasdaman uses relational databases to store raster tiles. Additionally, the

system can restructure tiles to speed-up frequent queries. It also allows irregular tiles, which are shaped to match the most frequent access patterns. A recent addition is the *Datacube* concept. Datacubes [Bau18; Bau+19; Bau+18] are a paradigm for providing earth observation data for spatial and temporal analysis. The basic idea is the unification into one large four-dimensional stack of arrays with common spatial and temporal resolutions, i.e., all rasters are reprojected to a common projection and resampled to one homogeneous pixel size. Access and combination of originally heterogeneous data is then much easier. However, resampling to reduce the resolution will result in loss of information.

Another array database is SciDB [Sto+11]. It uses tiles for data storage [SBW11; Sor+15] and a cache with prediction of future queries to enhances response times for interactive exploration [BCS16]. Planthaber *et al.* [PSF12] present an example for remote sensing raster data from MODIS. SciDB requires importing the data first. Then it allows ad-hoc analysis, e.g., calculation of the NDVI.

PostGIS [OH15] is an extension for Postgres, which enables extensive spatio-temporal functionality [SK91]. It allows raster and vector operations using SQL. However, raster data support still has limitations. Like SciDB and Rasdaman, fast access benefits from splitting the raster data into tiles.

While the presented array database systems provide methods for efficient access, importing raster data is often a complex task and requires a lot of processing time [WNA14]. Therefore, the NoDB concept [Ala+12], where data is read directly from files is supported by Rasdaman [BDM13] and SciDB [Xin+18]. ChronosDB [Rod19; Zal18] operates directly on files and provides support for distribution in computing clusters. While all three mentioned array databases support raster tiling for efficient access, another aspect are pyramids and aggregation. Wang *et al.* [WNA14] present different patterns for raster aggregation. Using them, they show that raster file access can outperform SciDB. Ladra *et al.* [LPS17] present a scalable and queryable compressed storage structure for raster data. They present a strategy similar to pyramids. Storing aggregates (min, max) of non-spatial attributes in each aggregation layer, they provide a compact data structure and allow efficient queries similar to Quad-Trees. GDAL (see Chapter 2.2.1) also supports tiles and pyramids. Pyramids generate an aggregated coarser views of the original data. Therefore, GDAL provides efficient data access for many applications.

3.1.2 Raster Processing

Processing the pixels of a raster is quite easy to perform in parallel. Thus GPUs, which are designed for this, are often used to speed up processing. Additionally, databases, as well as modern big data platforms, enable fast and parallel raster data processing.

Rasdaman supports parallel and distributed processing [Bau+18; Bau+16]. Splitting queries into sub-queries allows to distribute and execute them on multiple processing nodes. Additionally, if a request requires multiple tiles to be processed, they are also distributed over nodes. Rasdaman uses loop-fusion to move multiple calculations on single pixels into one loop iterating over a raster tile. The SciDB follows a similar approach. Multiple SciDB instances in a shared-nothing design employ clusters to process sub-queries on raster tiles [Rog+10; SBW11; Sor+15].

Big data frameworks also support processing spatio-temporal data. While most approaches focus on vector data, we focus on raster data processing. A comprehensive evaluation of popular tools, including Rasdaman, SciDB, and Spark is given by Hu *et al.* [Hu+18a]. The survey finds that all tools have pros and cons. Spark handles large volumes with stable resource consumption, while Rasdaman and SciDB provides sophisticated array operations.

Giachetta [Gia15] presents a *map-reduce* framework for processing large scale remote sensing data in Apache Hadoop. Raster data is partitioned into tiles (*map*) and processing is executed in parallel. Then, the tile results are merged (*reduce*) to generate a final result. ESRI, the company developing ArcGIS, also provides GIS tools for Hadoop[1].

Climatespark [Hu+18b], as well as the STARK framework [HBS19], allow raster data processing using the Spark framework [Zah+16]. Spark provides an abstraction of element collections, which are partitioned across the nodes of a cluster. This data model is called Resilient Distributed Dataset (RDD). STARK adds a RasterRDD and provides a web interface and allows queries formulated in SQL. STARK aims to process remote sensing data, which is too large for single node array database systems. Climatespark adds a ClimateRDD and operations to handle large spatio temporal raster data. It is designed to support scientists with on-demand processing and analytical capabilities using in-memory and distributed processing. GeoTrellis[2] is another raster processing solution for Spark. Similar to Climatespark, it allows spatio-temporal queries. It aims to provide

[1] `esri.github.io/gis-tools-for-hadoop`
[2] `geotrellis.io`

web-speed processing to enable an exploratory visualization and fast batch processing. Zurita-Milla *et al.* [Zur+17] calculate vegetation phenology for the US based on temperature and land surface with GeoTrellis. A special RDD for raster data enables processing in Spark, while queries are formulated in Scala, R or Python.

Another big data system is the Google Earth Engine[3] (GEE) [Gor+17]. It provides an online interface for spatio-temporal raster processing with large data repositories. Examples include detection of land cover change [SPC18], crop mapping [She+17], and wetland mapping [Hir+17]. Wilson and Jetz [WJ16] calculate cloud frequencies for individual months from the MODIS raster timeseries.

Adding more parallelism to raster processing can benefit from the array nature. Many-core hardware, e.g., GPUs can perform hundreds or thousands of threads at once. A performance evaluation of image processing algorithms on the GPU is given by Castaño-Díez *et al.* [Cas+08], which finds a high speed-up provided by GPUs. Early approaches to exploit GPUs for raster processing, e.g., in raster database systems like Rasdaman [BJS09] use the Open Graphics Library (OpenGL). OpenGL is designed for 3D graphics and modern languages for GPGPU processing are CUDA from Nvidia and OpenCL. OpenCL is the open standard for processing on SIMT hardware (see Chapter 2.4.3). Liu *et al.* [Liu+14] extended SciDB to use GPUs. They found a speed-up of 10 for selected tasks. Besides raster database systems, other approaches explore the application of GPUs for spatial data. The ORFEO toolbox [IC09] provides algorithms for processing of spatial data and [CMI11] extend it to use CUDA. They found speed-up of more than 100 for selected raster processing tasks. Sánchez *et al.* [Sán+15] use GPUs to speed up sub-pixel analysis of hyperspectral images, e.g., to generate vegetation maps and detect fires. The speed-up of the GPU implementation is again approximately 100. Zhang *et al.* [ZYG15] uses a hybrid combination of GPUs and CPUs in computing clusters to combine raster and vector data. They found that processing large-scale spatial data on such a setup gives speed-ups by orders of magnitude, in comparison with traditional techniques on single CPU cores. They also found that using GPU and CPU together can provide benefits. Zhang *et al.* [ZYG17] provide a web-based GIS where users are able to perform explorative queries, which are processed by GPUs to provide the required performance for interactive usage.

Remote sensing data often requires special handling of formats and calibration of raw data. This is usually done with special purpose tools, which also enable direct

[3] `earthengine.google.com`

access to the remote sensing data. Raspaud *et al.* [Ras+18] provide an open-source tool called *PyTroll* for accessing and processing earth observation raster data. It is implemented in Python and enables data scientists to directly access satellite data. There are also implementations for specific satellite and sensor platforms. Cermak *et al.* [CBD08] provide access and special purpose processing of MSG SEVIRI data. While a parallel implementation for processing MODIS data [Hua+16] is available to speedup processing times, such an implementation was not available for MSG SEVIRI data at the time of writing.

3.2 Cloud and Fog Detection

Detection of cloud cover and properties belongs to a broad research area. Stubenrauch *et al.* [Stu+13] presents a detailed overview of global cloud datasets from satellites. It states that remote sensing techniques using satellites provide the only way to generate global and long-term cloud datasets with spatial and temporal resolutions required for many applications. This review focuses on cloud datasets generated from LEOs and GEOs. One example is the 34 year cloud time series derived from *AVHRR* data, which is important for climate science [Kar+17].

Sensors on different satellites and orbits, like LEO and GEO, provide different resolutions. The impact of the resolution is discussed by Wielicki and Parker [WP92]. An overview on current and future LEOs and GEOs and their impact and application to derive weather and cloud parameters is given by Thies and Bendix [TB11]. Menzel *et al.* [Men+18] provides an overview of the history and developments of meteorological satellites. It also states the importance for applications like weather and climate analysis as well as weather prediction. An extended review of equipment and methodologies for cloud detection is provided by Tapakis and Charalambides [TC13]. It covers methods for differentiating clouds from other areas and to identify cloud types.

Active satellite instruments on CALIPSO, which uses LIght Detection And Ranging (LIDAR), and CloudSat, which uses RADAR, allow measuring the vertical distribution of clouds [Ove+11]. However, the majority of sensors on LEO and GEO platforms is passive and provides multispectral data in the infrared (IR) and visible (VIS) bands. Therefore, most cloud detection methods are designed to use spectral properties of a single channel or multiple channels to decide for each pixel whether it is covered by clouds, based on appropriate thresholds (e.g.,

Saunders and Kriebel [SK88]). However, dynamic characteristics such as changing SVA (the angle between satellite and the scanned surface) or changing solar irradiation (caused by the diurnal cycle of the sun) can cause challenges for threshold-based methods [CB07]. Dynamic threshold estimation allows deriving the relevant thresholds as a function of the contextual data, and therefore, adapts to changing characteristics. Another approach to detecting clouds relies on statistics computed from time series data. Based on these statistics, Schillings *et al.* [SMM04] introduced a clear sky state to identify cloud pixels from the differences between the clear sky (surface) and clouds. However, spectral properties cannot be clearly assigned to all kind of clouds in all cases, e.g., ice clouds and snow areas are challenging to separate [TB11]. Because of these deficiencies of the various methods, the available operational cloud mask products introduce a broad set of rules to combine multiple classification methods to obtain a decision for each pixel of multispectral satellite images.

For current systems like MODIS, cloud datasets [Ack+98] are routinely created. These products are also referred to as *operational*. Gradual improvements enhance the MODIS cloud datasets [Fre+08] [Bau+12]. Validation of cloud datasets use instruments on other satellites, e.g., the LIDAR data from *CALIOP* [Hol+08]. The ESA Cloud_cci project [Ste+17] provides a cloud dataset created from multiple satellites and sensors. However, the temporal resolution of LEOs is in the order of days. A setup of two LEOs such as Terra and Aqua, which carry MODIS, provide repetition cycles of 12 h outside the polar region.

In contrast, GEOs support high-frequent coverage, since a geostationary position can be used to continuously monitor the same area with a temporal resolution in the range of minutes. Therefore, only geostationary satellites can provide data with temporal resolutions that support real-time monitoring and creation of time series with high temporal resolutions [Stu+13]. For GEOs like MSG [Sch+02], Himawari [Bes+16] or GOES [Sch+05], which are used for weather prediction and warnings, there are cloud datasets with a temporal resolution in the order of minutes. The Cloud Mask (CMa) product from the CLAAS-2 dataset provides a classification of MSG pixels into cloud classes [Ben+17]. This dataset is introduced in Chapter 2.4.2. CLAAS-2 uses the full resolution of MSG SEVIRI. The validation of the dataset uses multiple other satellites [Ben+17]. Some applications, such as nowcasting, require timely cloud information. Creation of most of the datasets relies on a large set of rules and enabling nowcasting requires to run the CMa algorithm [CM 16] on a reduced resolution. To meet processing and delivery times required by nowcasting and other applications, very complex cloud detection algorithms are reengineered and reimplemented [HFS10].

Cloud masks using rules are composed of many tests for different cloud types, e.g.,

[DGF13]. Fog and Low Stratus (FLS) are examples of cloud classes. The investigation of FLS trends and patterns and their relation to climate change or air pollution relies on station data [KL16]. Synoptic observations about the state of the atmosphere are mostly provided by ground-based instruments at locations of special interest e.g., airports. The sparse distribution of them and therefore the sparse distribution of data prevents deriving spatial FLS products [Ben02]. Additionally, different measurement methods, frequencies, quality levels, and data gaps add additional challenges. At airports, Meteorological Aviation Routine Weather Reports (METAR) data is gathered and published. Difficulties are added as different time intervals are used and while large airports use sensors (e.g., cyclometer) there are still airports where the FLS situation is assessed manually [ETB18]. Remote sensing techniques are used to overcome these issues.

Different approaches allow classification of FLS from Low Earth Orbiter (LEO) data, including AVHRR [BB91; Mus+14] and MODIS [Sch+16]. However, the temporal resolution, depending on the repeat cycle of the spacecraft, renders LEO-based data problematic for real-time monitoring and prevents covering diurnal dynamics. Other approaches rely on Geostationary Earth Orbiters (GEO) like Meteosat Second Generation (MSG), which are designed to provide data for a fixed area with a high temporal resolution for now- and forecasting. Additionally, geostationary satellites, which are used for operational weather services provide time series of comparable data over long time intervals. Schemes to derive FLS products from MSG data have been developed for night [CB08] and day [CB07].

While originally developed for a study area covering Europe, they have been adapted to other areas [Cer12]. A combined day and night time series was created by Cermak *et al.* [Cer+09] for the winter months (Dec, Jan, Feb) from 2004 to 2008. A unification and efficient implementation, which is part of this thesis, allows to create a continuous ten year FLS climatography [Egl+17]. Further work to separate fog and low stratus [ETB18] use Machine Learning (ML) methods, in this case Random Forst (RF). Other recent developments train Multi Layer Perceptrons (MLP), e.g., for cloud detection [Tar+15], or use and compare multiple ML techniques [Mey+16]. Section 3.3 covers different approaches to apply ML for remote sensing.

3.3 Machine Learning

A multitude of studies covers Machine Learning (ML) approaches on remote sensing data. For cloud properties and precipitation retrievals, learning methods like

Random Forest (RF) and MLP, as well as deep learning with CNNs, show impressive results. A common strategy for classification of raster data using ML methods is a pixel-wise application of a trained model. Kühnlein *et al.* [Küh+14] use RF to model a relation between MSG SEVIRI data and rainfall measured by a the radar network of the German weather service [Bar+04]. First, the relevant cloud areas are identified with RF. Then, the cloud areas are separated into convective and advective areas to which rainrates are assigned to. Meyer *et al.* [Mey+16] use this setup to compare multiple ML methods, including RF, MLP, and Support Vector Machines (SVM) [LWA04]. The study finds that all selected methods perform well and show very small differences. A combination of RF, NN, and SVM can improve the classification of rainfall from MSG SEVIRI data for uncertain pixels [LA18]. In this context, Meyer *et al.* [Mey+17] found that spatial information is not relevant as predictor variable for shallow learning methods.

Egli *et al.* [ETB18] employ RF for FLS detection and separation. They state that spatial information in the form of handcrafted texture features are essential for the trained model. Feature engineering leads to spatial statistics that improve information significantly in this context of cloud detection. Texture features provides valuable information, since they are often distinct and less sensitive to atmospheric effects [Gu+89]. Cloud classification relying only on spatial properties [Ame+04; Gan+11] is also possible. Taravat *et al.* [Tar+15] train a MLP for cloud classification on MSG SEVIRI data with spatial statistics. This approach is further enhanced to enable short term prediction [Per+16]. However, identifying and handcrafting relevant spatial features is a cumbersome and time-consuming task. The main disadvantage of this process, however, is that it is impossible to identify all possible spatial patterns empirically. This leads to a high possibility of missing-out important ones.

A more recent class of ML methods are Convolutional Neural Networks (CNN) [LBH15; GBC16; He+17], which are highly suitable for object detection, image classification, and segmentation. The concept of CNNs is introduced in Chapter 2.5.2. Ball *et al.* [BAC17] provides an extended overview on the application of CNNs for remote sensing tasks, which range from ship detection to cloud detection, e.g. [Liu+19]. Similar to the threshold-based and shallow learning methods, cloud classification approaches using CNNs usually focus on deriving classifications for individual pixels, e.g., [Le +17]. Classification of ice and water pixels in satellite data is also possible [LT15], as well as classification of land-use per image [Cas+15] and per pixel [Mag+17]. Processing each pixel separately requires to execute a trained model for each pixel, which is more or less a sliding window moving over all pixels. Xie *et al.* [Xie+17] employ clustering methods as a post-processing step to group individual cloud pixels into cloud entities. Another strategy [Li+19] uses

super-pixels for pixel aggregation. Clustering all pixels into super-pixels reduces the number of objects, which are then classified by a CNN. Pelletier *et al.* [PWP18] add a temporal dimension to CNNs for change detection of land-use. The proposed TempCNN performs convolutions over a stacked time series and outperforms RF and other methods.

Another approach to classify all pixels is segmentation, where all pixels of a raster are classfied in a single step. CNN architectures for segmentation [NHH15; LSD15; BKC17] have been successfully applied to challenging problems in several areas. Examples are medical image segmentation [RPB15], segmenting buildings from high-resolution satellite RGB images [Xu+18; Wu+18], and the segmentation of woody areas from remote sensing data [FWC19]. The segmentation technique can incorporate higher level features and requires less redundant operations in comparison with single pixel methods. However, the best of our knowledge, there was no approach of segmentation of cloud areas in multi-spectral satellite data present prior to the approach presented in Chapter 7.

3.4 Interactive Visualization and Explorative Workflows

Visualization of spatio-temporal data usually focuses on spatial visualization. Peuquet [Peu99] discuss the aspect of time in GIS and finds that techniques for spatio-tempral data often focus on the spatial component. For the temporal component, GIS follow the snapshot approach and often leave time handling to the user. An overview on modeling spatio-temporal data is given by Pelekis *et al.* [Pel+04]. Erwig *et al.* [Erw+99] present modeling of moving objects. An algebra for spatio-temporal data was developed by Gebbert *et al.* [GLP19]. Time handling is also investigated for existing GIS systems [Raz12] and for event processing [KS05]. For raster time series, the snapshot approach is used by most applications.

Visualization and analysis of spatio-temporal data is usually performed in desktop GIS, e.g. QGis[4], ArcGIS[5], or GRASS[6]. They provide map views with support for geographical and projected CRS. Besides the map view, data properties and styles are displayed and editable. Desktop GIS provide large toolboxes with operators for creation and modification of spatio-temporal data, e.g. for spatial statistics [SJ10].

[4] www.qgis.com
[5] www.esri.com
[6] grass.org

For complex analysis tasks, popular desktop GIS provide Python bindings. This allows to use Python code as an operator inside a GIS or to use the functionality provided by a GIS in external code.

Spatio-temporal data, e.g., climate data [Ove+11], is very large and analysis requires efficient access and processing power [Yan+17]. Users from multiple domains require interactive exploration to identify relevant data and to create new insights. Therefore, web-based visualizations with database back-ends are used to enable access and visualization. Database systems like Rasdaman [Bau+18], SciDB [Sto+11] or PostGIS [OH15] provide services to manage data. Only Rasdaman implements the OGC protocols. The connection to the web is then provided by a middle-ware, e.g., GeoServer[7], which implements the relevant OGC standards (WMS, WFS, and WCS). Libraries for web-based maps like OpenLayers[8] or Leaflet[9] supply a two-dimensional map view for spatial raster and vector data. Visualization frameworks like Carto[10] add more interactive features like time-sliders, plots, and descriptions.

Using this technology stack, special purpose web-based applications, e.g. Map of Life [JMG12], LifeWatch [BL12] or Integrated Digitized Biocollections (iDigBio) [Bea14], provide web-based visualization for individual domains or topics. The Atlas of Living Australia [Bel11] uses a datacube approach to provide access to multiple datasets. New data is first ingested into the datacube and is then available for users. It also includes predefined analytic tools. While this allows interactive visualization and exploration, the functionality to compose data and operations is often limited to a predefined setting.

Desktop GIS often provide workflow composition for processing of complex tasks. Workflows enable reproduction and tracking of provenance, which is important for scientific work. Graser and Olaya [GO15] present the workflow implementation of QGIS, which allows to include external applications. Scientific workflow systems also use external tools to provide processing of spatio-temporal data. de Jesus *et al.* [dJes+12] model raster data processing as a workflow composed of web-services available via WCS in Tavera [Wol+13]. This requires the web-services to be available when an execution is triggered. Changes on the server side can cause differences between two executions. However, in all cases, users need to be familiar with every steps when the workflow is created. In addition to the processing, another advantage of workflows is to document complex processing in

[7] `geoserver.org`
[8] `openlayers.org`
[9] `leaflet.org`
[10] `www.carto.com`

an abstract manner. In particular, this is important for reasons of provenance and data lineage [IW09].

The Google Earth Engine (GEE) [Gor+17] offers large data repositories with an execution on large computing clusters and a web-based user interface. As mentioned above, the GEE requires defining operations as Python or JavaScript code. Visualization on a map is possible by using the Google Maps API. However, there is no visual component to compose workflows. GEE runs only in the Google cloud, which is sometimes not in agreement with legal aspects of data privacy. Another drawback of GEE is that users have to express all functionality on code. There is no simple visual interface for exploring data.

In conclusion, a mixture of all available approaches is required to allow users from all domains to access and use spatio-temporal data. A web-based system with integrated data management is required to provide access. Visual exploration is the key to enable non-programmers to combine and analyze the heterogeneous types of data. Time must be an integral dimension in the system to help users to explore temporal data over time. The visualization must be fast to enable an explorative usage, and workflows are required to model and reuse complex processing steps. Since non of the available systems provides all of these features, the VAT System [Bei+17c; Bei+17b; Aut+15b] was designed for this purpose. Chapter 8 presents it's components and features.

4

Data Access and Preprocessing

This chapter presents the processing of Meteosat Second Generation (MSG) raw data into physical values, which are required for further tasks. A focus of this chapter is the presentation of efficient and parallel raster processing modules. The basics of raster time series and remote sensing, e.g., satellite images, are covered in Chapter 2.2 and Chapter 2.3. Chapter 2.4.1 introduces MSG.

As discussed in Chapter 3, the amount and growth of remote sensing data requires approaches to tackle spatial big data challenges[Yan+17]. While different systems for processing spatio-temporal big data are developed, Data Scientists and domain experts also require a library approach, which enables them to access raster time series to develop new applications and classification methods. Therefore, efficient algorithms and parallel processing are required to achieve reasonable runtimes [Ma+15] for a single raster, as well as large time intervals. Related work (see Chapter 3.1.2) shows remarkable speed-up when using GPUs for raster processing.

The MSG satellites positioned at 0°N, 0°E generate new data every 15 min [Sch+02]. Since the MSG programs started in 2004, the available data represents a very long time series of raster data with a regular time step. At the time of writing, it covers approximately 500,000 scenes, which are about 110 MB per scene when compressed. With a total size of approximately 40 TB, this dataset is an excellent example of a raster time series produced by satellites. As introduced in Chapter 2.4.1, the main instrument SEVIRI on MSG has 11 channels covering both, the VISible (VIS) and InfraRed (IR) spectrum. Each of these channels is sensitive to specific absorption or emission bands in the electromagnetic spectrum. Therefore, channels are often also referenced as *bands*. To summarize all rasters of the various channels at a point in time, we use the term *scene*.

The analysis of situations visible in satellite data, as well as the development of new products, requires meaningful physical values to derive rule-based classifications or to identify relevant features for shallow and deep machine learning.

However, the sensors on satellites usually produce unit-less raw data, which represents the count of radiation received by the sensor. Therefore, before we can perform any analysis or generate any product, the raw data requires preprocessing.

To enable processing of large time series, an efficient implementation of data access, calibration, and transformation is of utmost importance. This chapter, therefore, presents such an implementation, which uses parallel processing on CPUs and GPUs as well as efficient methods for data access and concurrent processing of multiple processing steps. This enables the application of different analysis and machine learning methods in the following chapters, including the provision of data for a ten-year FLS climatology and deep learning for cloud detection.

This chapter is divided into four sections. First, we discuss the access to MSG raw data and metadata in Section 4.1. Section 4.2 presents the theory and conversion operations used to transform MSG raw data into physical values. Section 4.3 introduces the modular processing chain for MSG data. This includes parallel processing with OpenCL on GPUs or many-core CPUs and efficient data access. Finally, we evaluate the performance of the presented implementation in Section 4.4.

4.1 Accessing MSG Level 1.5 Data

SEVIRI generates raw raster data and metadata, which contain scan conditions and data descriptions. Both are stored in a specific data format called High Rate Information Transmission (HRIT) [EUM15]. Note, that there is also a Low Rate Information Transmission (LRIT). Both are sometimes summarized as XRIT. The HRIT data format was specially developed for geostationary weather satellites by the Coordination Group of Meteorological Satellites[1] (CGMS). The documentation of the HRIT format also specifies the GEOstationary Satellite view (GEOS) projection, which is the projection used to map the raster data to coordinates. This specific projection is relevant for the correct calibration of the raw data and is essential to combine SEVIRI data with data in different projections.

The raw data produced by SEVIRI is called Level 1.0 data and can contain different radiometric and geometric effects, which are corrected by EUMETSAT [EUM10b].

[1] `www.cgms-info.org`

The result is the so-called Level 1.5 data, which is then offered in multiple formats, including the HRIT format.

In comparison to common formats like HDF or GeoTIFF, accessing the HRIT format is not trivial. The special wavelet compression of the HRIT files requires the use of decompression code provided by EUMETSAT [EUM11]. The *Wavelet Transform Software* allows building a binary application *xRITDecompress*, which generates uncompressed copies of HRIT-files on the filesystem. The layout of the uncompressed files is according to the MSG Level 1.5 Image Data Format Description [EUM10b]. This information allows developing a parser for the uncompressed files. This approach is followed by the Python library PyTroll[2], which is developed for accessing earth observation satellite data [Ras+18].

Creating a copy of the raw data on a file system can be avoided by using the popular Geographic Data Abstraction Library (GDAL). GDAL supports common formats by default, and users can enable support for MSG SEVIRI data in a custom build. This requires to enable the MSG driver[3] and including the source code of the *Wavelet Transform Software*. To skip the copy on the filesystem, the GDAL library includes the wavelet decompression code. This permits to load the uncompressed data directly into main memory for metadata and raster access. Therefore, we have decided to build the processing with GDAL as data access layer. Additionally, GDAL can be used very easily in different programming languages, including Python and C/C++, which is a bonus when developing in a heterogeneous environment.

In the HRIT format, the 11 VIS and IR channels, as well as the HRV channel, are stored individually. Each of the 11 VIS and IR channels are additionally divided into 8 files, which contain data blocks. Each block covers a fixed area of the earth's surface. The blocks are arranged along the scan direction from south to north and are 3712×464 pixels in size. This is also the case for the HRV channel, which has three times the spatial resolution and is therefore divided into 24 blocks. GDAL has the advantage that it optimizes the block access and therefore I/O operations [GDA17]. For a study area covering Europe, which intersects only two northern blocks, only these blocks are unpacked, and the corresponding pixels are loaded into memory. The blocks are visualized as overly for an MSG SEVIRI scene in Figure 4.1. The Figure shows an RGB composite generated from the MSG SEVIRI full disk on the left-hand side. Europe covers only a small area in the image. On the right-hand side, the zoom to Europe indicates that only two data blocks are required for this area.

[2] `pytroll.github.io`
[3] `gdal.org/drivers/raster/msg.html`

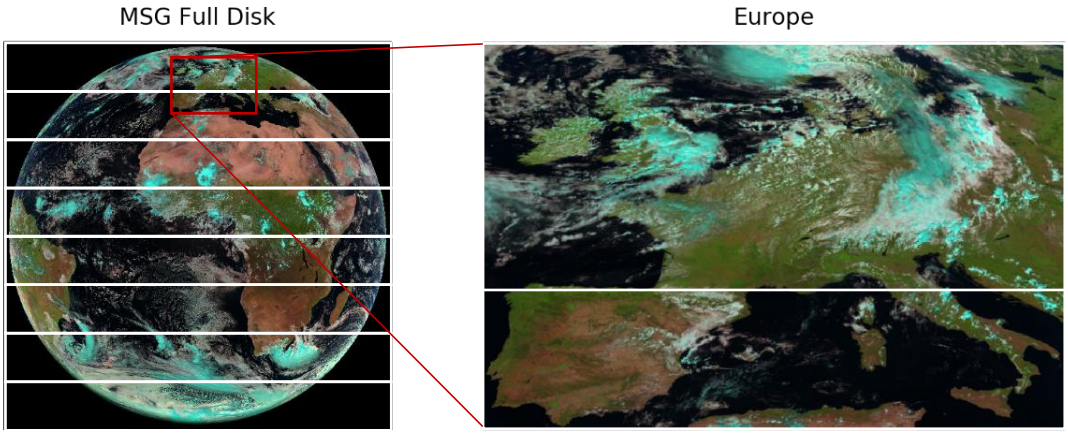


Figure 4.1: The MSG SEVIRI data is stored as individual blocks. The block structure of the HRIT files is represented as overlay on the MSG RGB composite on the image on the left. The right image shows that only two data blocks are relevant for Europe.

The HRIT format has a fixed header where information like the start and end times of a scan, and calibration information is stored. At the time of this work, it was not possible to access the required metadata from the HRIT-files in GDAL. Therefore, GDAL's MSG driver was extended by us to allow the required metadata access. Besides the addition of access to the metadata[4], we also fixed a bug in the calculation of the coordinate reference for MSG data[5].

Each SEVIRI channel requires a transformation into physical values. This transformation uses calibration parameters called *offset* and *slope*, which are stored in the channel metadata. The following code Listing 4.1 shows an example of how to load metadata and all pixels of a specific channel from an HRIT-file using GDAL in Python.

Listing 4.1: Reading MSG Data with GDAL in Python

```
1
2 from osgeo import gdal
3 import datetime
4 # variable parts of HRIT filename
5 msg_name = 'MSG4'
6 channel = 'IR_108'
7 date_time = datetime(2018, 08, 01, 12, 15) # 2018-08-01 12:15
8 date_str = datetime.strftime(date_time, '%Y%m%d%H%M')
9
```

[4] github.com/OSGeo/gdal/commit/5542866b2d9c5067aca9206797277be037daa7b5
[5] github.com/OSGeo/gdal/commit/36e918fac56504c69ecc75aea5b9cc27650d170d

```
10 # build HRIT filename
11 filename = "H-000-"+msg_name+"__-"+msg_name+"________-"+channel+"
       ___-000001___-"+date_str+"-C_"
12
13 # open dataset
14 dataset = gdal.Open(filename)
15
16 # Get the raster. It is always the first one (MSG/HRIT).
17 band = dataset.GetRasterBand(1) # GDAL starts at 1
18
19 # get msg specific metadata
20 raw_metadata = band.GetMetadata("msg")
21 metadata = {
22   'calibration_offset': float(raw_metadata['calibration_offset']),
23   'calibration_slope': float(raw_metadata['calibration_slope'])
24 }
25
26 # read the full disk pixels as 2d numpy array
27 xoff = 0
28 yoff = 0
29 win_xsize = 3712
30 win_ysize = 3712
31 data = band.ReadAsArray(xoff, yoff, win_xsize, win_ysize)
```

While HRIT stores the data of a channel in eight different files, GDAL only requires a single filename. The other filenames are derived automatically. In Line 5 to 8 the variable fields of an HRIT filename are prepared, and a complete filename is built in Line 11. Line 14 creates a new dataset object from the specified file, and in Line 14 the first raster in the dataset object is accessed. While GDAL supports multiple channels per dataset, the MSG driver does not use this concept because all channels have individual files. In Line 20, we access the channel's metadata and parse the relevant calibration parameters in the following lines. Finally, in Line 27 to 30, the bounds of the raster to read are specified. In this case, the complete raster is requested, and Line 31 triggers the creation of a new two-dimensional NumPy array containing the pixel values.

4.2 Calibration of MSG SEVIRI Data

Figure 4.2 shows the data calibration steps for MSG SEVIRI data. First, the raw data is scaled to radiance values. Then, the radiance values are transformed to brightness temperatures or reflectance values, depending on the channel type.

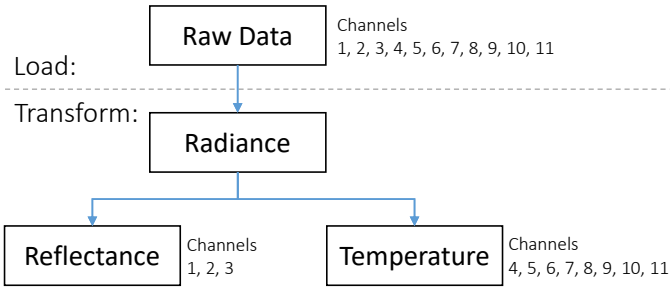


Figure 4.2: The processing flow for MSG SEVIRI images. First, the data loaded from HRIT files represents raw radiation count information. The second step is the transformation into calibrated radiance values for all channels. The third step transforms radiance values into reflectance for Channel 1–3 and brightness temperatures for the other channels.

The first preprocessing step transforms raw values into radiance L_λ, where λ denotes the central wavelength of a channel. This step involves the channel specific *slope* and *offset* parameters from the HRIT metadata

$$L_\lambda = \text{raw}_\lambda \cdot \text{slope}_\lambda + \text{offset}_\lambda.$$

This is the basic transformation which is followed by further specific steps for VIS and IR channels.

IR channels are mapped to brightness temperatures (T_λ), using look-up tables [EUM12b] or translation formulas [EUM12c]. Since each SEVIRI instrument of the MSG program has different characteristics, a look-up table *Lut* is available for each combination of IR channel central wavelength (λ) and satellite (S). Assuming that the look-up table has a direct mapping for each value, the transformation requires a three-dimensional look-up

$$T_\lambda = \text{Lut}[S][\lambda][L_\lambda].$$

For VIS channels, L_λ values are transformed into reflectance values R_λ, which represent the reflected solar radiation. This bidirectional reflectance factor is calculated using constants measured for each satellite and wavelength as

$$R_\lambda = \frac{\pi \cdot L_\lambda \cdot d^2(t)}{I_\lambda}$$

where d is the distance between the earth and the sun at time t represented in AU and I_λ is the irradiance of the channel at 1 AU [EUM12a].

This way, R_λ is initially calculated using a constant solar irradiation for each pixel, assuming the Solar Zenith Angle (SZA) to be 0°. See Figure 2.8 for a visualization of the SZA. However, the SZA changes with the diurnal cycle and even for a single scene, each pixel has a separate angle due to the curvature of the Earth. To correct R_λ, each pixel value is divided by the cosinus of its actual SZA

$$R_\lambda = \frac{\pi \cdot L_\lambda \cdot d^2(t)}{I_\lambda \cdot \cos(\mathrm{SZA}(t, x))},$$

where $\mathrm{SZA}(t, x)$ is the Solar Zenith Angle in radians at time t and a pixel's location x [EUM12a; Cer06].

Figure 4.3 presents the SZA and the solar azimuth angle for the MSG SEVIRI full disk on 2018-08-10 12:00:00 UTC in the top row. Both angles indicate the position of the sun for each pixel. The lower row shows the reflectance calculated with the individual SZA of each pixel on the left-hand side. On the right-hand side, the reflectance was computed without the SZA. Visual inspection confirms that the impact of this effect is enormous. The corrected image appears homogeneous while the other one shows a gradient.

4.3 Efficient preprocessing

To enable processing of the large MSG SEVIRI time series (40 TB and growing), this section focuses on efficient methods and parallel processing. The presented code is implemented as a Python library. Code, which is executed only once per raster is implemented directly with the Python standard library and uses NumPy arrays [vdWCV11] for two-dimensional raster handling. We used GDAL to access the raster data (Listing 4.1) and implemented the calibration methods described in Chapter 4.2. Operations applied to all pixels are implemented using OpenCL. As shown in Chapter 2.4.3, OpenCL is ideal for implementing parallel operations for raster data. OpenCL provides a heterogeneous computing model that permits applications to run on multi-core CPUs and on GPUs.

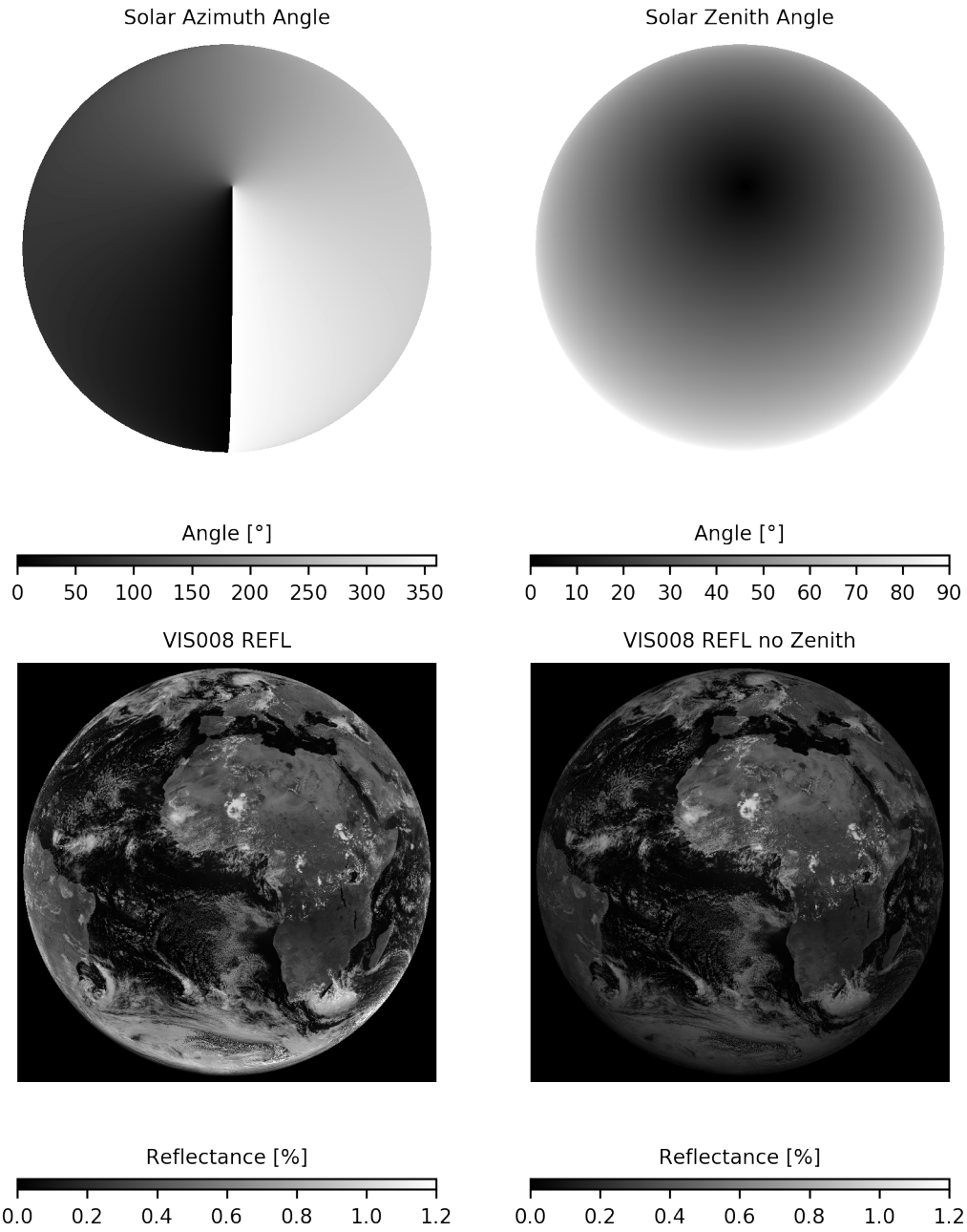


Figure 4.3: The top row shows the Solar Azimuth and the SZA for the scene created at 2018-08-10 12:00:00 UTC. The bottom row shows the reflectance of channel VIS_008. On the left, the calibration was applied with normalization using the SZA. On the right, the SZA was ignored. The corrected image appears homogeneous while the other one shows a gradient with bright clouds in the center and darker clouds near the borders.

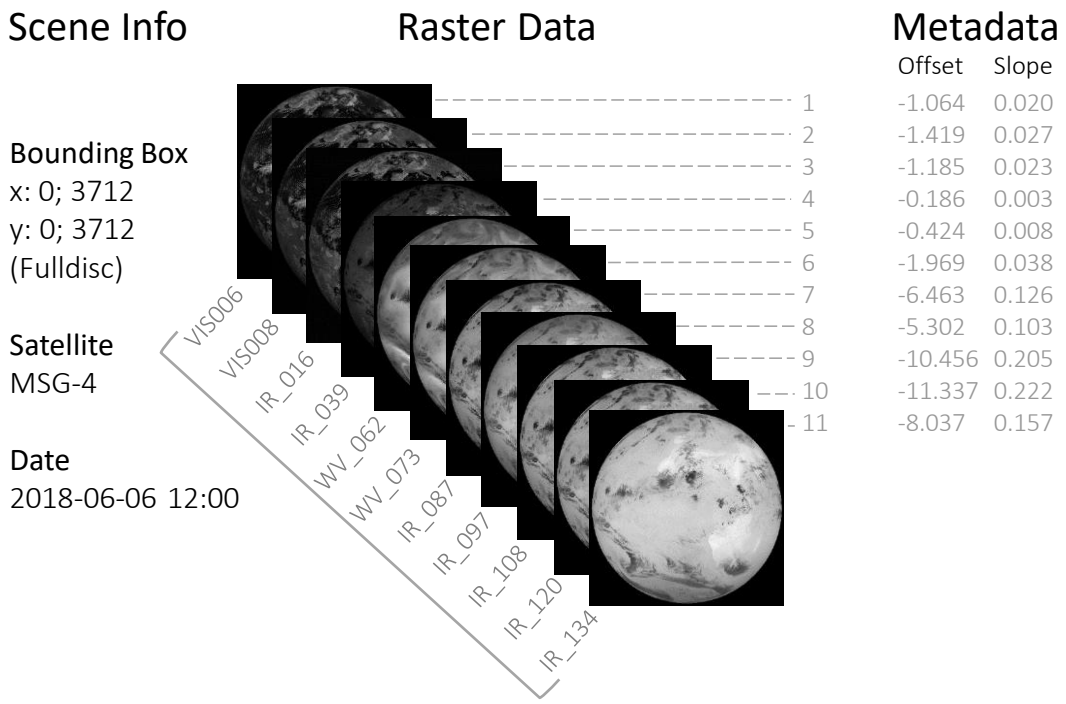


Figure 4.4: Structure of a scene object. The scene carries global information like creation time and covered area. Each two-dimensional raster is linked by name to the corresponding SEVIRI channel and relevant metadata is stored for each channel.

We implemented two modules, one for data loading and one for data transformation. Instead of processing single channels, both modules process entire MSG SEVIRI scenes. Figure 4.4 presents the structure of a scene object. We allow scenes to be restricted by a *Bounding Box* to specify a study area, e.g., Europe. A scene carries additional global information, including *Satellite* identification and the creation *Date* and time. A dictionary stores the two-dimensional raster of each channel, which are referenced by name. For all channels, we store the available metadata information. The processing steps either replace channels or add new raster with a corresponding name to the dictionary. One example is the SZA-raster, which is added as the "Zenith" channel.

MSG Raster Loading

The first module handles data loading. It uses the code from Listing 4.1 to load data from HRIT-files with GDAL. First, the module generates the filename of the selected channels at a given date and time. The filename is passed to GDAL, which

loads all pixels inside a bounding box for all requested channels. The corresponding metadata, especially the slope and offset values for each channel, are also fetched and stored. A scene object is created for all loaded channels, as well as the relevant metadata.

Since the loading of each channel involves CPU time for decompression, a sequential loading of channels does not saturate I/O. Therefore, we have extended the data loading module in a way that a dedicated thread is used for each channel. This implementation enables to load and queue one or more channels from the storage device while others are decompressed.

MSG Raster Calibration

The second module transforms the MSG SEVIRI scenes. As a first step, we transform all channels into radiance values. This uses the slope and offset values from each channel's metadata. The following code snippet represents a simplified OpenCL code for this task. The OpenCL Kernel is started with a two-dimensional global work-size corresponding to the dimensions of each raster.

Listing 4.2: Transforming raw data to radiance values with OpenCL

```
1 kernel void rawToRadianceKernel(global short *in, global float *
      out, float offset, float slope) {
2  int x_size = get_global_size(0);
3  int x = get_global_id(0);
4  int y = get_global_id(1);
5
6  //pixel position in linear memory
7  int g = y * x_size + x;
8
9  //transformation
10  out[g] = in[g] * slope + offset;
11 }
```

Using the *get_global_** methods from Line 2 and 3, a thread gets information about its location within the global work-size. For pixel-wise operations, this is identical to the pixel location in the raster image. In Line 7, the pixel location is mapped to the one-dimensional address *g*, which represents the location in global memory where the pixel value is stored. In Line 10, the raw pixel data is read from *in* at address *g*. The raw value is then transformed into radiance and written to *out* at the same address.

VIS channels are further transformed into reflectance values. For each pixel, the individual SZA is needed to correct the calculated reflectance value. To determine the SZA for each pixel, the creation time of the scene, and the location of each pixel is required. To optimize this step, all global information, like the sun position measured as ascension and declination, the Greenwich Mean Sidereal Time (GMST), and the Earth Sun Distance (ESD), are calculated once for each scene. This is possible because only the creation time of a scene is required to calculate these values. In order to calculate the reflectance values for each pixel, the transformation of a pixel position into latitude and longitude coordinates is required. Listing 4.3 presents an OpenCL Kernel that calculates the location dependent SZA for each pixel and transforms the input radiances into reflectance values.

Listing 4.3: Transformation of radiance values into reflectance with OpenCL

```
1 kernel void radToReflKernel(
2     global float *in, global float *out,
3     double scale_x, double scale_y,
4     double origin_x, double origin_y,
5     double viewAngleFactor,
6     double sub_lon,
7     double GMSRT,
8     double ascension, double declination,
9     double ETSR, double ESD
10 ) {
11   int xsize = get_global_size(0);
12   int x = get_global_id(0);
13   int y = get_global_id(1);
14   int g = y*x_size + x;
15
16   double2 xy = (int2)(x, y);
17   double2 origin2 = (double2)(origin_x, origin_y);
18   double2 scale2 = (double2)(scale_x, scale_y);
19   double2 geosPosition = xy * scale2 + origin2;
20
21   double2 viewAngle = geosPosition * viewAngleFactor;
22   double2 latLon = viewAngleToLatLon(viewAngle, sub_lon);
23   double zenith = calcZenith(GMST, ascension, declination, latLon)
       ;
24
25   float radiance = in[g];
26   out[gid] = radiance * (ESD * ESD) / (ETSR * cos(radians(zenith))
       );
27 }
```

The *get_global_** methods from Line 2, 3, and 4 provide information about the

location of a thread within the global work-size. In Line 7 to 9, vector operations are used to handle two-dimensional values, like the pixel coordinate or the raster origin. First, the pixel coordinates, as well as scale and origin, are mapped to two-dimensional vectors. In Line 10, these parameters are used to calculate the position of a pixel in the GEOS projection [EUM10a]. In Line 12, a predefined factor (viewAngleFactor) transforms the GEOS coordinates into the SVA. This is the angle between satellite, sub-satellite point (SSP) and a pixel. We convert the viewing angle into each pixel's Lat/Lon coordinate using the GEOS projection definition from EUMETSAT [EUM10a] in Line 13. This conversion is also implemented using OpenCL. In Line 14, the SZA for the pixel is calculated based on the GMST, the sun position, and the pixels coordinate. The pixel radiance value is read from *in* at address *g* in Line 16. We calculate the reflectance as described in Section 4.2 and write the result to *out*.

IR channels are further transformed into brightness temperatures (T). However, using the already transformed radiance values to look-up the corresponding temperature from a table [EUM12b] is not efficient. The radiance values are floating-point values with a satellite and channel-dependent range. The temperature is a linear spaced value between 100 K and 350 K. Therefore, a naive approach requires a search for the best matching reflectance in the table.

The SEVIRI raw data is stored as 10-bit integers, which permits only 1024 different values. This allows a very efficient approach to avoid the look-up for all pixel values. First, the radiance and corresponding brightness temperatures are computed for all 1024 possible values in a channel. Since this is a very small workload, this step is done on the CPU. Then, we copy the generated look-up table to the global memory of the OpenCL device along with the raw value raster. A simple OpenCL Kernel, that uses this mapping to transform raw pixel values directly into temperatures, is given in Listing 4.4.

Listing 4.4: Look-up of brightness temperature values with raw data values using OpenCL

```
1 kernel void rawToBbtKernel(global short *in, global float *out,
      global float *lut) {
2   int x_size = get_global_size(0);
3   int x = get_global_id(0);
4   int y = get_global_id(1);
5
6   //pixel position in linear memory
7   int g = y * x_size + x;
8
9   //look-up
10  out[g] = lut[in[g]];
```

```
11 }
```

Again, the *get_global_** methods from Line 2, 3, and 4 provide information about a thread's location within the global work-size. In Line 7, we map the pixel location to the address *g* in linear memory. In Line 10, the raw pixel data is read from *in* at address *g*. It is then used as an address for the provided lookup table. A resulting temperature value is written into *out* at address *g*. This approach avoids redundant computations for re-occurring raw values.

4.4 Performance Evaluation

This section presents the performance evaluation of the MSG processing modules. We conducted our experiments on an AMD Ryzen 7 2700X eight-core processor (16 threads, 32GB RAM) and an AMD Radeon RX 580 GPU (8GB VRAM). For both devices, OpenCL implementations based on the LLVM compiler infrastructure [LA04] are available: The Portable Computing Language (POCL) [Jää+15] for the CPU and the AMD ROCm Platform [ROC18] for the GPU. All experiments use a Samsung 970 EVO NVMe solid-state disk to load and write data.

We compare five different experimental setups. All setups processed the same set of 600 scenes, which were loaded from the SSD. First, we execute data loading and preprocessing sequentially. A single thread is used for data loading, and the OpenCL execution is carried out on the CPU with a single CU. Second, the OpenCL execution is allowed to use all CPU threads, which results in 16 CUs. Third, we run the OpenCL execution on the AMD Radeon RX 580 GPU. The fourth setup uses multithreaded data loading (8 threads), and the OpenCL code is executed on a GPU. The execution is adapted to run data loading and calibration steps in parallel, for the fifth setup.

Figure 4.5 shows the runtimes of the various processing steps for different configurations. The first step is the extraction from the raw data from a tar-file. This step avoids the reuse of cached data by different test runs.

The baseline is *cpu01*, which uses sequential loading and processing and is limited to a single CPU CU for OpenCL. Executing loading and calibration sequentially takes approximately 9 s per scene.

Configuration *cpu16* utilizes all available CPU threads and CPU CUs for OpenCL. The performance of the calibration step is not improved with multiple CPU cores.

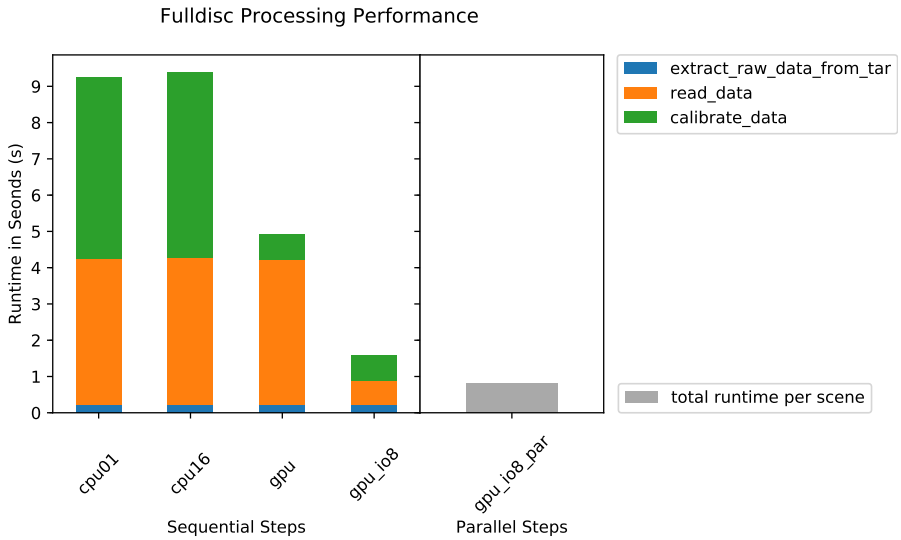


Figure 4.5: The runtimes for the five test setups. The first two setups use only the CPU with 1 and 16 cores. The third setup uses the GPU and the fourth setup adds parallel I/O. The fifth setup runs all steps in parallel.

While this result appears unusual, the array operations of the calibration steps are probably vectorized by the OpenCL runtime and therefore show no difference.

Configuration *gpu* performs all OpenCL operations on the GPU. This heterogeneous approach combines CPU (I/O) and GPU (calibration). The calibration time is reduced significantly (5.10 s to 0.69 s). The GPU is much faster than the CPU for this task, which uses many trigonometric operations to calculate the Lat/Lon coordinates and the SZA for each pixel.

Configuration *gpu_io8* uses multiple threads to load the channels of each scene in parallel and to reduce waiting on I/O. The resulting runtime (0.61 s instead of 4.66 s) is a significant improvement.

While a sequential execution allows processing a full disk scene in approximately 1.6 s, overlapping multiple scenes can reduce the average processing time, which is essential for time series. The overlapping execution allows a processing time per scene, bounded by the slowest processing step. For setup *gpu_io8_par*, loading and processing steps are executed concurrently. This allows to load a new scene while the previous one is still in the calibration step. This reduces the processing time per scene to 0.825 s.

4.5 Conclusion

This chapter presents the steps required to access and calibrate MSG SEVIRI data. The presented processing modules provides efficient and parallel processing of both steps. This implementation enables the usage of a large raster time series like the MSG SEVIRI data, which includes approximately 500,000 scenes. We also added support to load MSG data stored in other formats, e.g., HDF. The following chapters contain different threshold and machine learning-based approaches, where the large MSG SEVIRI time series is used. The presented calibration or preprocessing implementation is essential to provide the requied data.

5

Fast Fog and Low Stratus Detection

This chapter focuses on the detection of Fog and Low Stratus (FLS) from the MSG SEVIRI time series. There are many different aspects of FLS, and while some of them are beneficial, others can cause dangerous situations. FLS influence climatic processes, ecological systems, and many aspects of human life [Gul+07]. By influencing the radiation balance of the atmosphere, FLS acts as a modifier in the climatic system [VYvO09]. In some, otherwise arid, ecosystems, FLS provide the main water supply [BEB05; VB99]. There are, however, also downsides of FLS. Inversion situations can lead to smog, with adverse effects on human and animal health [NHN01]. Additionally, the reduced solar irradiation, during FLS events, limits solar power production, which can destabilize energy grids [Köh+17]. The most direct impact of FLS on human life concerns land, air, and sea traffic. Reduced visibility leads to hazardous situations, and more frequent or severe accidents are observed [BEK11]. For these cases, it would not only be beneficial to know about the distribution of FLS but also to detect the development and occurrences of FLS in real-time. FLS information with high temporal and spatial resolution could facilitate instant reactions to FLS events, e.g., issuing warnings or rerouting traffic. As introduced in Chapter 2.4.1, MSG SEVIRI generates a new scene every 15 min since 2004. A time series of more than ten years, where ten years include approximately 350,400 scenes, is also a valuable source for climate research. Norris *et al.* [Nor+16] report that long-term satellite data indicate evidence for climate change.

There are multiple approaches to detect cloud pixels using classification rules derived from expert knowledge e.g., the Cloud Mask (CMa) of the CLAAS-2 dataset [Ben+17] introduced in Chapter 2.4.2. However, the classification of FLS and its separation from other cloud pixels is a more complex task. We present related work for FLS classification in Section 3.2. Cermak *et al.* [Cer+09] created a combined day and night FLS classification time series for the winter months (Dec, Jan, Feb) from the years 2004 to 2008.

A long-term time series with high spatial and temporal coverage can provide new

insights into the distribution of FLS over time and space. Therefore, we developed an FLS detection based on the methods used by Cermak *et al.* [Cer+09]. It extends the modular preprocessing chain, presented in Chapter 4 and adds modules for the steps required to detect FLS. In this chapter, we present our FLS detection approach in detail. The methods from the day and night FLS detection schemes by Cermak and Bendix [CB07; CB08] are unified to use consistent partitioning strategies and are enhanced for fast and efficient processing. Our approach supports flexible study areas of arbitrary size, while the previous methods [CB07; CB08] were originally designed for small and fixed-size study areas, e.g., Europe. Similar to the previous chapter, OpenCL [SGS10] is used to enable parallelism on CPUs and GPUs with the same program code. This, as well as efficient algorithms for histogram creation and a sweep-line-based histogram analysis, enable processing ten years of MSG SEVIRI data for Europe in under 2.5 days on commodity hardware. Overall, we successfully generated a full ten-year FLS climatology for Europe [Egl+17] by applying our adopted implementation.

This chapter is structured as follows. First, an explanation of FLS situations is given. In Section 5.2 we recapture the properties of MSG SEVIRI data and the FLS detection concepts [Cer06; CB07; CB08]. Section 5.3 covers our unification and enhancements of the FLS detection concepts and focuses the details of our implementation. Processing results and performance, as well as our main use-case, the *FLS climatography*, are discussed in Section 5.4.

5.1 Fog and Low Stratus

An observer on the ground can easily distinguish between Fog and Low Stratus; however, in the raster data produced by a sensor on a satellite like MSG SEVIRI both appear similar. The Glossary of Meterology[1] contains the international definition of fog. Fog consists of droplets in the atmosphere and fulfills two additional conditions: First, fog differs from clouds in that it must touch the ground and second, fog has a visibility (line of sight) below 1000 m. The International Cloud Atlas[2] defines stratus clouds as a grey cloud layer with a uniform base. Low stratus is a stratus layer near the earth surface, which can develop from fog, among other causes, when the fog layer is lifted from the ground.

Figure 5.1 shows a sketch of a landscape with different fog and cloud situations. Colored circles mark four situations, and lines in the same color represent the view of a satellite-based sensor like MSG SEVIRI. The **first** situation is colored orange.

[1] `glossary.ametsoc.org/wiki/Fog`
[2] `cloudatlas.wmo.int/stratus-st.html`

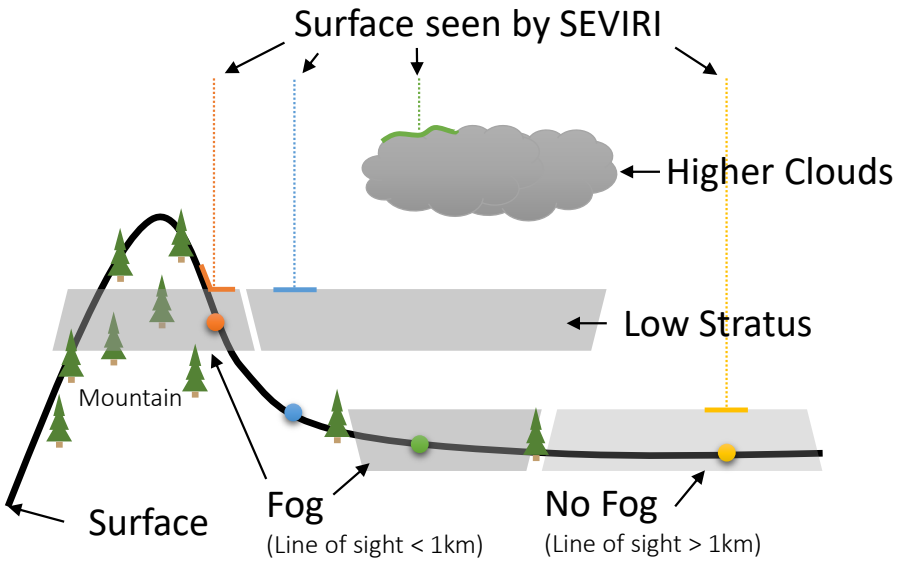


Figure 5.1: A conceptual representation of fog and low stratus situations. Individual colors mark four situations: Orange indicates satiation where a stratus cloud causes fog on an elevated surface. Blue shows that FLS seen by a satellite is not always fog on the ground. Green visualizes that MSG SEVIRI always sees the top surface, which can hide fog situations. Finally, the yellow situation is no fog by definition since the visibility changes when compared to the green situation.

This is a fog situation at a mountain slope with visibility below 100 m. A satellite looking at this position would show the fog surface. The top of the mountain would emerge from the fog surface in the satellite image. However, a cloud entity touching the ground creates the fog. The same entity is present in the **second** situation, which is marked blue. At this location, no fog is present. However, there is a low stratus cloud above this location, and the satellite image shows the same surface as in the first situation. It is therefore difficult to distinguish between fog and low stratus in the MSG SEVIRI raster data. The **third** situation is marked green. There is a higher cloud layer above the low stratus layer and an additional fog layer. In this case, sensors like MSG SEVIRI show the surface of the higher clouds. Thus, there may be fog or low stratus situations that cannot be detected in the satellite data. The **fourth** situation is marked yellow and shows a situation in which the visibility is only slightly above the 1000 m of the fog definition, e.g., 1001 m. While this is no fog by definition, the satellite data will show a fog-like situation. Therefore, the subject of the presented method is a combined classification of FLS.

5.2 Data and Methods

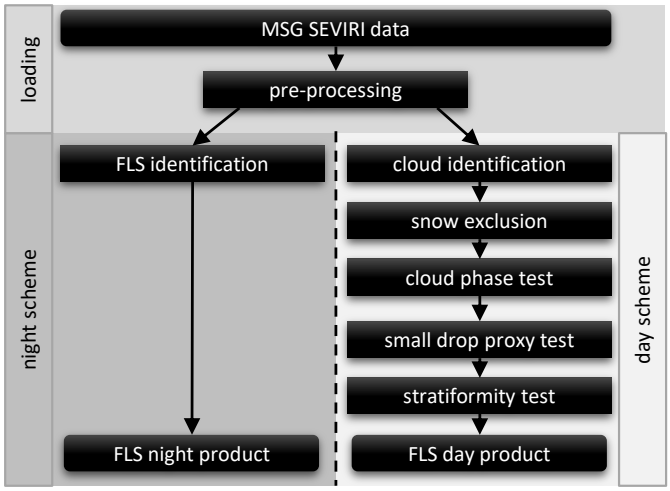


Figure 5.2: The processing flows of the day and night schemes for FLS detection combined in one diagramm.

In this section, we present the concepts of FLS detection by Cermak and Bendix [CB07; CB08] and revisit the MSG SEVIRI data. Figure 5.2 shows an overview of the day and night FLS detection schemes. First, we presents the night scheme and the concepts for deriving FLS thresholds using histogram analysis. Then, the more complex FLS detection at daytime is recaptured. The data loading and preprocessing steps are identical for both schemes and use the concepts introduced in Chapter 4.2.

5.2.1 Meteosat Second Generation Data

Meteosat Second Generation (MSG) satellites provide meteorological data since 2004 from a geostationary orbit with a sub-satellite point (SSP) at 0°N 0°E. SEVIRI generates raster data covering a hemisphere, including Europe, Africa, and the Atlantic Ocean with a repeat cycle of 15 min. Eleven main channels are centered at wavelengths λ from 0.6 µm to 13.2 µm, covering the VISual (VIS) and InfraRed (IR) spectrum. Each raster image has a size of 3712×3712 pixels and a resolution of 3 km × 3 km at the SSP. Channels 1 to 3 cover the VIS spectrum, while channels 4 to 11 cover the IR spectrum. The raw SEVIRI data is loaded and transformed using the processing chain presented in Chapter 4.

For this chapter, the following two properties of the MSG SEVIR data are essential First, the distance between a pixel and MSG depends on the SVA (see Chapter 2.4.1). The channel at 3.9 µm is sensitive to CO_2 absorption. With a larger distance between sensor and pixel, the amount of CO_2 increases. Therefore, the SVA influences the measured pixel values.

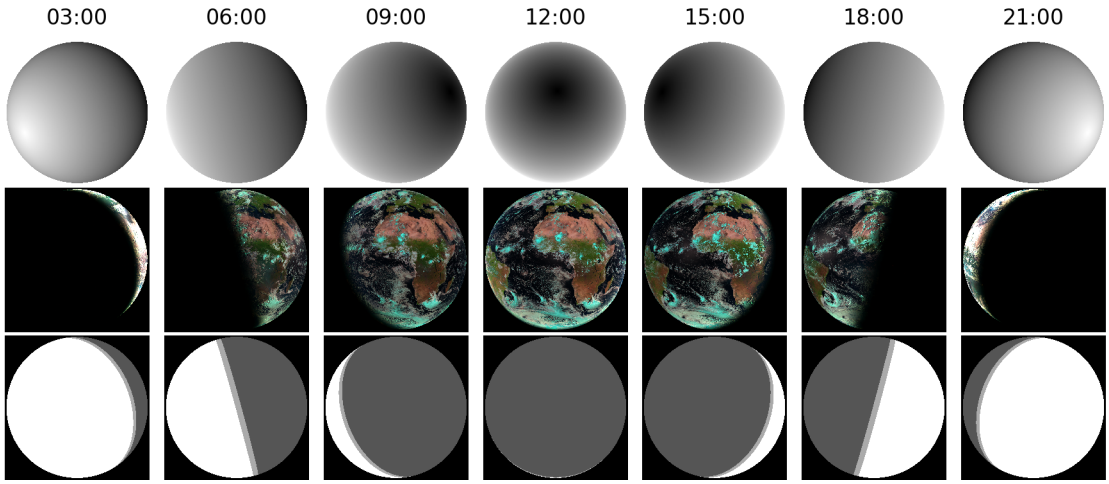


Figure 5.3: The diurnal progression in 3 hour steps for August 10, 2018. The top row shows the SZA, the second row shows the RGB composit generated from channels 1 to 3, and the bottom row shows the classification into day, night, and the ±2° twilight buffer.

Solar irradiance causes the second effect. While the influence of the solar irradiance is evident for the channels 1 to 3, it is also influencing the signal in Channel 4 at 3.9 µm. While the VIS channels are dark at night, Channel 4 has different characteristics. To remove errors caused by mixed conditions, a ±2° buffer is applied. This effect is visualized in Figure 5.3. The figure shows three time series of the diurnal progression in three-hour steps for August 10, 2018. The first row presents the SZA, where black indicates an angle of 0° and white 180°. The second row depicts the RGB composite, which visualizes that RGB channels are dark at night. A twilight zone between dark and illuminated areas causes problems when using Channel 4. In these areas, the solar and thermal signals overlap. Therefore, pixels are classified into day and night, while twilight pixels with mixed conditions are skipped. This is indicated in the third row. The rules used to classify pixels are

$$\text{SZA} > 92^\circ \rightarrow \text{night},$$
$$\text{SZA} \leq 88^\circ \rightarrow \text{day}.$$

5.2.2 FLS Detection

The FLS detection methods by Cermak [Cer06] and Cermak and Bendix [CB07; CB08] employ a combination of the IR channels centered at 3.9 µm ($IR_{3.9}$) and 10.8 µm ($IR_{10.8}$). For day and night a dynamic threshold (Thr) is derived to classify pixels based on the channel difference $\Delta T_{\text{diff}} = T_{IR10.8} - T_{IR3.9}$. If $\Delta T_{\text{diff}} > Thr_{\text{night}}$ at night, a pixel is covered by FLS. However, at daytime the only clouds are classified by $\Delta T_{\text{diff}} < Thr_{\text{day}}$.

At night, the emissive properties of small droplets (SD) with a radius $r \approx 4\,\mu\text{m}$ and large droplets (LD) ($r \approx 11\,\mu\text{m}$) allow the classification of pixels as covered by SD, LD or as cloud-free. At daytime, $IR_{3.9}$ contains reflected solar irradiation which requires multiple steps to identify SD. Therefore, the detection schemes, are separated for day and night based on the SZA as presented in the previous section. A combination of the day and night methods is required to produce an FLS product covering the whole day. First, we present the FLS detection approach at nighttime. The used methods are then extended for the more complex daytime approach.

FLS Detection at Nighttime

At night, the emissive properties of droplets are used to separate SD and LD as presented by Hunt [Hun73].

At a wavelength of $\lambda \approx 4\,\mu\text{m}$ the emissivity of SD is lower than at $\lambda \approx 11\,\mu\text{m}$, causing $T_{3.9} < T_{10.8}$ [Hun73; CB07]. At the same time, larger droplets show a higher emissivity in the 3.9 µm channel ($IR_{3.9}$), leading to $T_{3.9} > T_{10.8}$.

Figure 5.4 shows an idealized histogram of ΔT_{diff} values where we assume no interfering effects. The histogram presents a value range from −15 Kelvin (K) to 10 K with a bin size of 1/3 K. Cloud-free pixels have similar $T_{3.9}$ and $T_{10.8}$ values which cause a *clear peak* where $\Delta T_{\text{diff}} \approx 0$. This is indicated in Figure 5.4 as a green line. SD, where $T_{3.9}$ is smaller than $T_{10.8}$, are located in histogram bins on the right of the clear peak at $\Delta T_{diff} > 0$. LD, where $T_{3.9}$ is larger than $T_{10.8}$, are

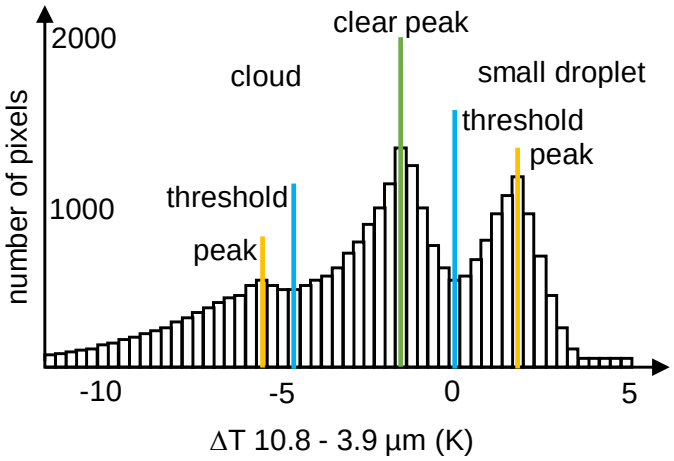


Figure 5.4: Idealized histogram of ΔT_{diff} showing peaks and derived thresholds for SD and cloud pixels.

located in histogram bins on the left of the clear peak [CB07; CB08]. This leads to the following classification:

$$T_{3.9} < T_{10.8} \rightarrow \text{SD},$$
$$T_{3.9} \approx T_{10.8} \rightarrow \text{clear},$$
$$T_{3.9} > T_{10.8} \rightarrow \text{LD}.$$

Figure 5.4 also shows visible peaks for SD and cloud pixels. Both are indicated by yellow lines. This is caused by the positive difference between the two channels on the right-hand side of the clear peak, and the negative differences on the left-hand side. The histogram allows deriving two thresholds, indicated by blue lines in Figure 5.4, which separate SD and LD from clear pixels.

Based on this observation, rules to select thresholds were derived [CB07; CB08]. If a peak is present at a value higher than reasonable for clear peaks, it is labeled as an SD peak, and the minimum between both peaks is selected as an Upper Threshold (UT). If a peak exists below the clear peak, it is labeled as the LD peak, and the minimum between both peaks is selected as the Lower Threshold (LT). If no significant SD or cloud peak is present, the flanks of the clear peak are analyzed to select a threshold where the slope drops significantly.

The threshold detection at night by Cermak and Bendix [CB07] is designed to handle effects caused by the SVA. Channel $IR_{3.9}$ is sensitive to CO_2 absorption, which

increases with the distance between pixel and SEVIRI. Therefore, histograms are created for pixel groups based on the SVA. Pixels are grouped in steps of 0.5° until a minimal amount of pixels is inserted into each histogram. This is done to ensure that the histogram is representative. For each group of pixels, the histogram is analyzed to derive the threshold for SD (UT). Then, a linear regression is derived between SVAs and detected thresholds. This function is then applied to classify each pixel as either clear or covered by SD, which is characteristic for FLS [CB07].

FLS Detection at Daytime

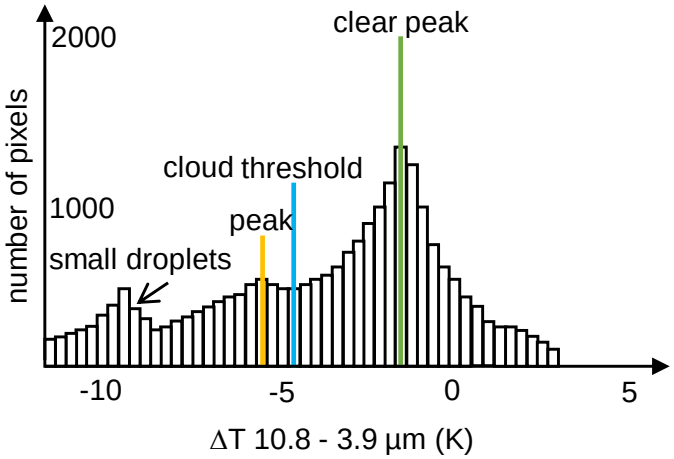


Figure 5.5: Idealized histogram of ΔT_{diff} at daytime. It indicates the peaks and derived thresholds. The solar component in SEVIRI Channel 4 at 3.9 µm causes SD and LD to be mixed.

The FLS detection at daytime uses a histogram-based threshold detection, similar to the nighttime approach. However, the solar component of $IR_{3.9}$ presents a challenging problem for the threshold estimation at daytime. To solve this, Cermak [Cer06] and Cermak and Bendix [CB08] present an extended solution based on multiple refinement steps. Reflected solar irradiation is the dominant component of the measured radiance $L_{3.9}$ and therefore heavily influences the resulting $T_{3.9}$. It causes $T_{3.9} > T_{10.8}$ for all cloud pixels which shifts ΔT_{diff} for all clouds including FLS to negative values. This effect is represented in Figure 5.5. The LT is now a cloud threshold, which separates SD and LD from cloud-free pixels [Cer06; CB08]. Therefore, multiple steps, as shown in Figure 5.2, are applied to refine the cloud-covered pixels. The result of the refinement is a filtered cloud mask where all no-FLS pixels are removed.

The first step is the **cloud pixel detection**, which uses a histogram-based threshold estimation similar to the threshold estimation at night. For this approach, the LT is derived from a single histogram which contains all pixels of a scene. Using the detected LT, all pixels are marked either as cloud candidates or as cloud-free, which leads to

$$\Delta T_{\text{diff}} \leq \text{LT} \rightarrow \text{cloud candidate},$$
$$\Delta T_{\text{diff}} > \text{LT} \rightarrow \text{cloud-free}.$$

Using the LT threshold, it is possible that pixels covered by snow or ice clouds are selected as cloud candidates. Ice clouds are clouds where droplets are in the ice phase. Therefore, further **refinement of cloud candidate pixels** is required. First, the potential snow pixels are identified using a threshold for $T_{10.8}$ and a second threshold for reflectance ($R_{0.8}$), leading to

$$T_{10.8} < 256\,\text{K} \text{ and } R_{0.8} > 0.11 \rightarrow \text{snow candidate}.$$

If both tests are true, a pixel is classified as potentially snow-covered. Then, snow-covered pixels are identified based on the Normalized Difference Snow Index (NDSI) [Doz89]. Pixels are classified as snow and are removed from the cloud candidates where the following condition is satisfied

$$\text{NDSI} = \frac{R_{0.6} - R_{1.6}}{R_{0.6} + R_{1.6}} < 0.4.$$

Since snow (and ice) pixels are now removed, the candidate pixels are further refined to warm cloud candidate pixels. Therefore, the next step handles ice clouds, which are clouds with droplets in the ice phase. Pixels covered by ice clouds are also removed from the warm cloud candidate pixels based on multiple tests: First, all pixels with a brightness temperature too cold to be in the water phase are identified as ice clouds and are removed from the warm cloud candidate pixels

$$T_{10.8} < 230\,\text{K} \rightarrow \text{ice cloud}.$$

Then, the remaining cloud candidates are refined based on the difference between $T_{8.7}$ and $T_{12.0}$, which is smaller than 0.65 K for ice clouds [SAM94]

$$T_{12.0} - T_{8.7} \leq 0.65\,\mathrm{K} \rightarrow \text{ice cloud.}$$

Additional tests are performed to assure that all cirrus clouds are included [Cer06; CB08]. The emissivity difference at $IR_{10.8}$ and $IR_{12.0}$ is utilized to detect cirrus clouds based on interpolated thresholds from a look-up table [SK88]. Additionally, a second test for cirrus clouds based on the difference of $T_{8.7}$ and $T_{10.8}$ is applied [WSS98]

$$T_{8.7} - T_{10.8} > 0\,\mathrm{K} \rightarrow \text{cirrus cloud.}$$

SD and LD are still mixed. However, FLS is considered to contain SD. Therefore, the **Small Drop Proxy test** (SDP) removes pixels with LD based on different signal characteristics of large and SD. At $\lambda = 3.9\,\mu\mathrm{m}$ SD have a higher reflectance than LD or clear pixels. At the same time, clear pixels have a higher thermal signal than LD. Therefore, the average $T_{3.9}$ value of the clear pixels (CP) is calculated over a selected area. This average $T_{3.9}$ value is then used as a threshold to remove pixels covered by LD [Cer06; CB08]

$$\text{average}(T_{3.9}(\text{CP})) < T_{3.9}(\text{SD}) \rightarrow \text{small droplet.}$$

Since the channel at $3.9\,\mu\mathrm{m}$ is sensitive to CO_2 absorption, SVA influences the measured pixel values. Therefore, pixels are grouped based on the distance to the sensor, which correlates with SVA to derive a threshold for SD. Since the study area covered by Cermak [Cer06] and Cermak and Bendix [CB08] is focused on Europe, image lines approximate the distance. The underlying assumption is that the pixels of each line have a similar distance to the sensor. For each group of pixels, the average $T_{3.9}$ value is calculated and serves as the threshold to reject LD pixels.

Recall that the remaining candidate pixels have SDs and are in the water phase. The last step uses the characteristic stratiform surface of stratus clouds to identify FLS. Therefore, the remaining candidate pixels are aggregated into FLS candidate entities. As stratiform surfaces are characteristic for FLS, the cloud top height of each entity is tested. The cloud top temperature at $T_{10.8}$ is utilized as a proxy for the cloud top height. An entity is classified as FLS if the standard deviation of its pixels is below $2\,\mathrm{K}$.

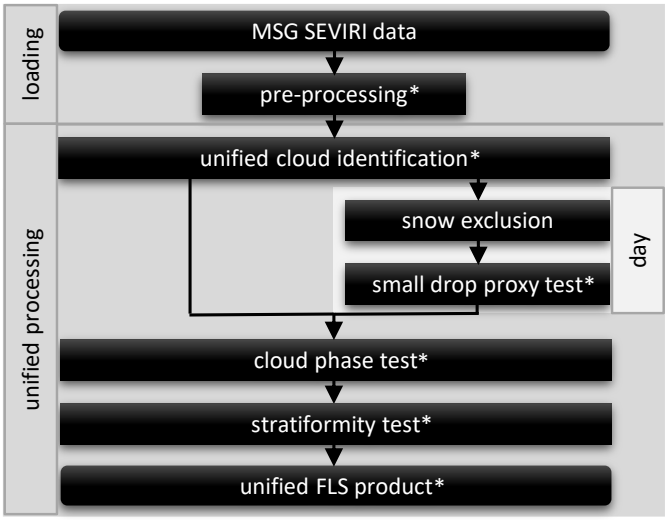


Figure 5.6: Unified processing flow used in the presented implementation for both, day and night processing. Steps marked with a star (*) are unified or adapted from the previous approaches. This is discussed in detail Section 5.3.2.

5.3 Unified and Enhanced FLS Detection

In this section, we present our enhanced and unified FLS detection approach and discuss its implementation. Figure 5.6 shows the unified processing flow. While the FLS detection schemes presented by Cermak [Cer06] and Cermak and Bendix [CB07; CB08], target FLS detection at night and day as separate tasks, we unified the processing steps into a single scheme whenever possible. Steps marked with a star (*) are unified or adapted.

In the original methods, dynamic threshold values are calculated in different ways and sometimes with very coarse granularity as described in Section 5.2.2. A consistent partitioning of the raster data into small tiles replaces the different approaches of the original methods. For each tile both, the uniform solar and the satellites viewing angle-dependent thresholds, are generated. With this strategy, we identify cloud pixels in one homogeneous step for day and night. The same partitioning is applied for the SDP test to unmix large and SD at daytime. The partitioning also enables parallel processing of tiles, which, in combination with efficient algorithms and raster processing on GPUs with OpenCL, enables the desired processing speed.

5.3.1 Implementation Design

The FLS detection is implemented as extension of the preprocessing modules presented in Chapter 4. For the FLS detection, additional channels representing the SVA and the SZA are added for each scene. Similar to the preprocessing modules, all FLS detection steps are implemented in a modular fashion and include modules for FLS/cloud detection, snow detection, refinement, and interpolation.

The implementation of the additional modules uses OpenCL [SGS10] for pixel-wise raster processing as introduced in Chapter 2.4.3. Complex operations, such as histogram generation and the determination of homogeneous surfaces in the stratiformity test, also use OpenCL.

5.3.2 FLS Detection

FLS detection is applied to the preprocessed data as shown in Figure 5.6. We adapted the histogram generation and analysis, the SDP test, and the stratiformity test as all three use spatial information. We apply the cloud phase test for day and night, and the snow test only for daytime. Each step produces an intermediate raster. All rasters created by the tests are combined to produce the final FLS raster, which stores the classification per pixel.

First, the unified threshold detection is applied to generate thresholds for SD at night and cloud pixels at daytime. To unify the day and night methods [Cer06; CB07; CB08], we partition the raster into tiles as shown in Figure 5.7. This harmonizes the different partitioning approaches of the original schemes and also allows to process the tiles in parallel. When selecting a small tile size, the range of SVA values per tile is limited. This also fulfills the same purpose as the pixel grouping of the original method. In contrast to the original schemes, where the SZA is only used implicitly to separate day and night, the partitioning into tiles also groups pixels with similar the SZA ranges.

Histogram Generation and Analysis

The histogram-based threshold estimation (Section 5.2.2) is unified into one module to generate FLS candidate pixels for day and night.

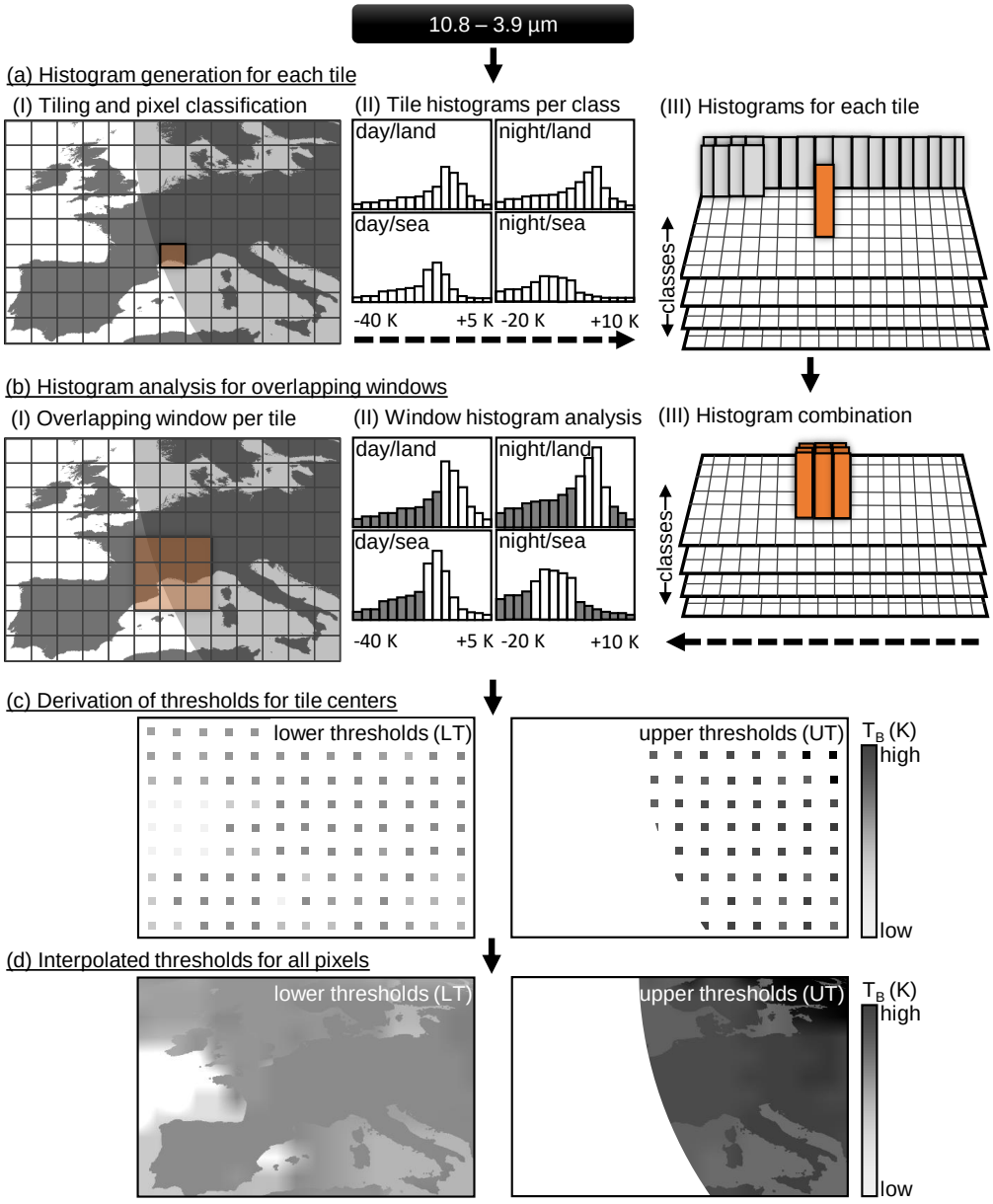


Figure 5.7: The unified histogram generation and analysis consists of three steps. (a) Histogram generation for each tile (highlighted). For each class of land-sea and light mask combination (day-land, day-sea, night-land, and night-sea) histograms are generated. (b) For each tile, histograms are combined with those of adjacent tiles to form overlapping windows. We perform the analysis for combined histograms for each tile and mask. Gray bars denote values interpreted as 'cloudy'. (c) Derived thresholds are assigned to tile centers. We compute lower thresholds for all mask combinations, upper thresholds only for night mask combinations. (d) Tile thresholds are interpolated over all pixels.

First, we generate histograms for each tile. Then, the histograms are analyzed using a sweep-line algorithm to detect thresholds for each tile. Using the tile thresholds derived from the histograms, thresholds are interpolated for each pixel.

Parallel Histogram Generation The parallel histogram generation consists of four steps, as shown in Figure 5.7. In the first step, the study area is partitioned into a grid of tiles as shown in Figure 5.7 (a)(I). A tile covers multiple pixels, e.g., 48×48 pixels. As in the day and night schemes, the pixels of each tile are classified as day or night using the SZA and as land or sea using a land-sea mask derived from the SRTM elevation dataset. This avoids multiple cloud-free peaks in a histogram as both aspects can cause significant differences.

We generate histograms and store them for each tile. This is done for the pixels of the four classes *land-day*, *land-night*, *sea-day*, and *sea-night*. Figure 5.7(a)(II) shows the histograms generated for the tile highlighted in Figure 5.7(a)(I). The generated histograms are stored, as shown in Figure 5.7(a)(III). For each class a separate grid stores the histograms for each tile.

Before the histogram-based threshold detection is applied, we have to ensure that the histograms contain enough data to be representative. In a second step (Figure 5.7 (b)), a window is created for each tile that overlaps its neighborhood (e.g., 3×3 tiles). The histograms within each window are combined and assigned to the original tile. This adds more spatial information per tile as they are not entirely isolated from each other. Figure 5.7(b)(I) shows the window for the tile, highlighted in Figure 5.7(a)(I). The previously generated tile histogram (Figure 5.7(a)(III)) is combined with the histogram of its neighborhood (e.g., 3×3 tiles) as shown in Figure 5.7(b)(III).

A naive approach would generate histograms separately for each class and an overlapping window. However, we create histograms for all classes at once. Tiles are created and combined in parallel, based on an OpenCL image histogram algorithm [Gas+13]. Following the pattern of the image histogram, a Kernel generates local histograms in local memory for small blocks, e.g., 8×8 pixels. Then, we store the histograms in global memory. A second Kernel combines them to the actual tile or window size.

Sweep-line based Threshold Detection For each tile, the combined histograms are analyzed using the method described in Section 5.2. From the combined histograms (5.7(b)(III)), thresholds for each class are derived as shown in Figure 5.7(b)(II). To efficiently analyze the combined histograms, we implemented a

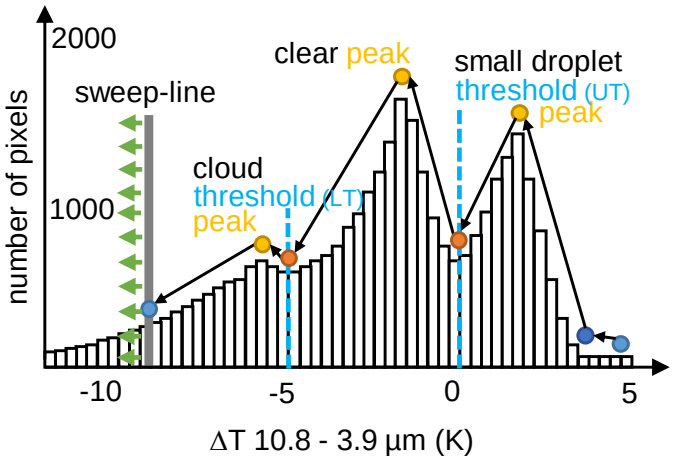


Figure 5.8: The sweep-line histogram analysis is performed per tile. Markers are placed at peaks, valleys, and changing slopes to derive thresholds.

sweep-line method to scan each histogram only once from right to left. Due to the independence of histograms, the analysis can be performed in parallel.

The sweep-line approach is visualized in Figure 5.8. To guarantee that all peaks are detected, the sweep-line starts at a value large enough. Significant slope changes, peaks, and minima are stored as checkpoints. At each new checkpoints, the previous peaks and minima are re-evaluated if they are the cloud, SD, or cloud-free peak or the corresponding minima. The checkpoints are updated accordingly during the scan, as illustrated in Figure 5.8. Then, UT and LT are derived using the rules described in Section 5.2.

Threshold Interpolation For each class, we generate a grid equal to the pixel grid. However, the tile thresholds only represent the center pixels, as shown in Figure 5.7(c). Night classes use UT to separate SD from other pixels and day classes use LT to separate cloud-covered pixels from clear pixels.

To remove over- or underestimations, tile thresholds are validated using the standard deviation of their tile neighborhood (e.g., 3×3 tiles). If a threshold exceeds a confidence range, in this case, two times the standard deviation, it is replaced by the average value of the tiles neighborhood.

Applying the tile threshold to each pixel within a tile creates sharp edges at tile borders, which does not reflect the continuous nature of the atmosphere. To solve this, thresholds are interpolated between tile centers to create individual

thresholds for each pixel, as shown in Figure 5.7 (d). The correction and the bilinear interpolation steps are implemented as individual OpenCL Kernels and are applied for each pixel.

The resulting thresholds allow to classify ΔT_{diff} pixels as covered by SD at night and to identify cloud candidate pixels at day. The SDP test is applied to separate large and SD at daytime.

Small Droplet Proxy Test

The SDP test [Cer06; CB08] approximates the distance between pixels and SEVIRI by image lines as described in Section 5.2.2. A partitioning into tiles keeps the approximation of the distance and provides a homogeneous partitioning through the whole processing. The implementation is similar to the histogram generation: We split the study area into tiles and derive the SD threshold for $T_{3.9}$ by calculating the average value of cloud-free pixels per tile. The threshold of each tile is then used to filter the cloud-covered pixels. Resulting pixels are considered as covered by SD and are therefore FLS candidates.

Stratiformity Test and FLS Mask Creation

Stratiformity is a characteristic property of FLS surfaces [Cer06; CB08]. $T_{10.8}$ serves as a proxy for the cloud top height. In contrast to the original scheme, however, individual pixels are considered instead of spatial entities. The standard deviation of all FLS candidate pixels within a neighborhood (e.g., 3×3 pixel) is calculated and a threshold (again twice the standard deviation) is applied to determine non-stratus pixels to be removed. The computation of the standard deviation of each pixel environment can be performed in parallel and is implemented using OpenCL. All stratiform FLS candidates are now referred to as FLS pixels.

5.4 Results and Discussion

In the following, we present the results for an example scene and discuss the processing performance. Finally, we revisit our main use-case, the ten-year FLS climatology. The FLS results and the climatology are created for Europe since the climatology also served to validate the produced results.

5.4.1 Processing Example

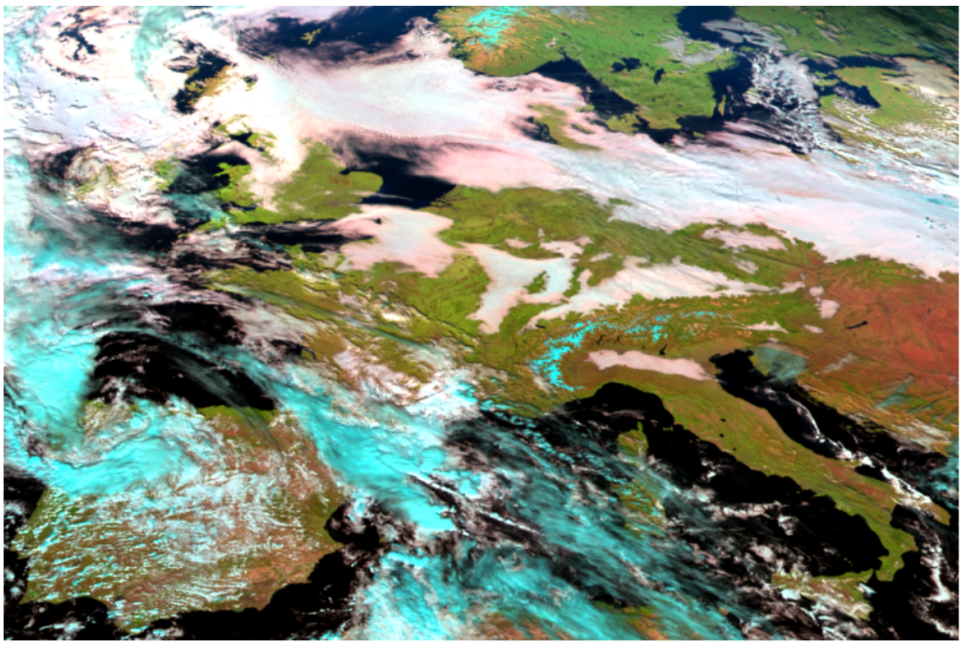


Figure 5.9: RGB composite of two visible channels and one near infrared channel for the study area on 2011-11-15 12:00 UTC.

Figure 5.9 shows the RGB composite from the VIS MSG channels at 2011-11-15 12:00 UTC for our main study area covering Europe. The processing of this scene produces an FLS mask, as shown in Figure 5.10. A large FLS patch is detected in the northern parts of the domain, stretching from the UK to the Baltics and Poland. Besides many smaller FLS patches, FLS also cover the Po valley, the Swiss plateau, and the Upper Rhine valley. On the other hand, mountain ranges like the Alps, the Black Forrest, and the Vosges are high enough to reach through the FLS layers. This is a very typical weather situation in the fall and winter months in Europe: Anticyclonic conditions result in stable atmospheric layering and radiative cooling processes close to the surface, especially within river valleys, result in condensation and subsequent formation of FLS layers. These processes do not influence areas with higher elevations, which are consequently FLS-free.

One enhancement of our implementation is the partitioning into tiles. For each tile, a cloud or SD threshold is derived using the histogram analysis presented in Section 5.3.2. FLS thresholds (UT) and cloud thresholds (LT) are visualized in

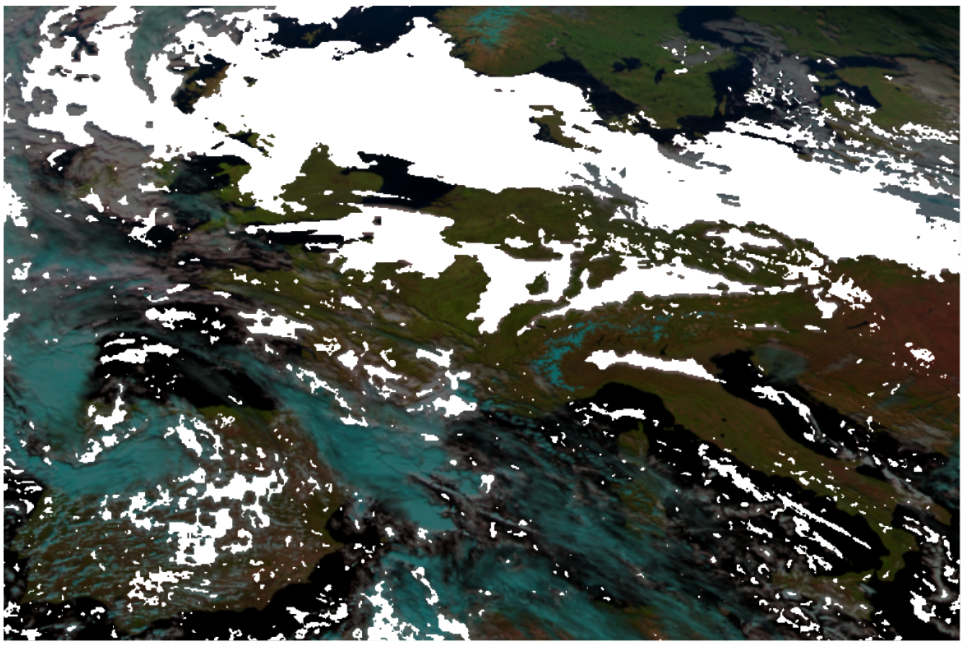


Figure 5.10: FLS mask (white overlay) for the study area on 2011-11-15 12:00 UTC.

Figure 5.7 (d). The interpolated thresholds show a gradient along the SVA and clear differentiation between land and sea.

5.4.2 Processing Performance

We conduct experiments on an AMD Ryzen 7 2700X eight-core processor (16 threads, 32GB RAM) and an AMD Radeon RX 580 GPU (8GB VRAM). For both devices, OpenCL implementations based on the LLVM compiler infrastructure [LA04] are available: The Portable Computing Language (POCL) [Jää+15] for the CPU and the AMD ROCm Platform [ROC18] for the GPU. All experiments use a Samsung 970 EVO NVMe solid-state disk to load and write data.

The average processing time for a single scene, e.g., the one presented in Figure 5.9, is 0.6 s. This low processing time is particularly important for large time series. As data is produced in 15 min intervals, processing ten years takes 2.5 days. Moreover, multiple machines would enable linear speed-up.

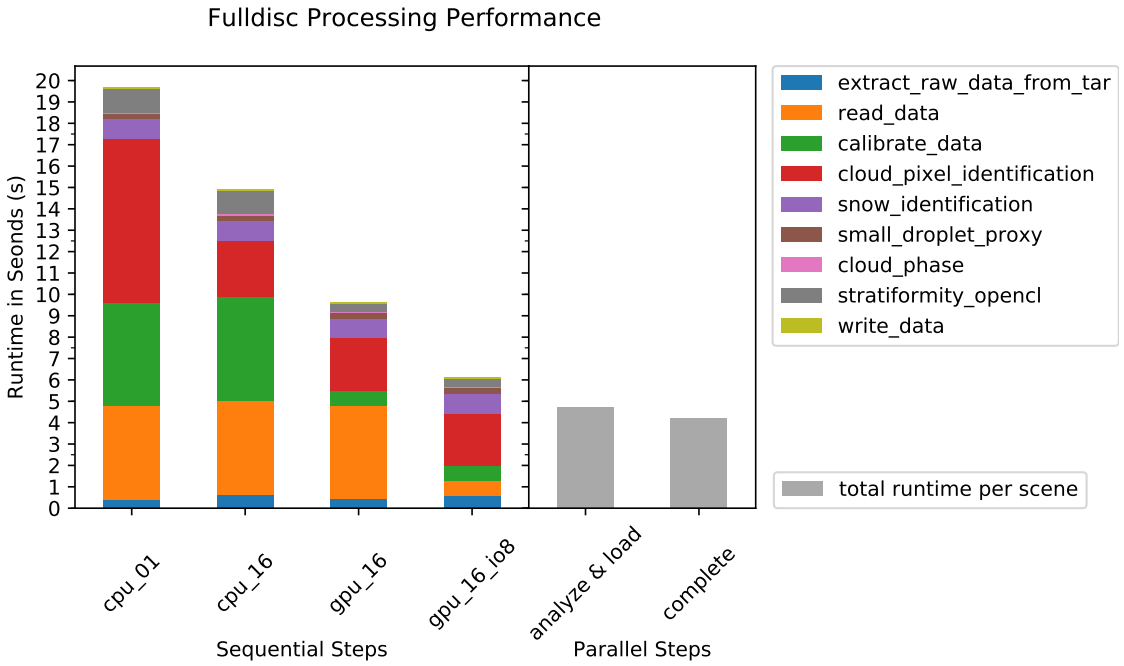


Figure 5.11: Processing performance for full disk images. Sequential execution strategies show the influence of parallelism, GPU and I/O. Overlapping execution of multiple steps enables even shorter runtimes.

To investigate the effects of parallelism, GPU acceleration and I/O, a test set of 600 full-disk images from 2011 was selected. While we leave the operationalization and validation of full disk scenes to our future work (e.g., to consider effects like sun-glint), the larger amount of data is ideally suited for a benchmark.

Figure 5.11 shows the runtimes of the various processing steps for different configurations. The baseline is *cpu_01*, limited to a single CPU thread and a single CPU CU for OpenCL. Sequentially running all processing steps takes approximately 20 s per scene.

Configuration *cpu_16* utilizes all available CPU threads and CPU CUs for OpenCL. The cloud pixel identification time (red) is significantly reduced from 7.68 s to 2.66 s. Our parallel sweep-line-based implementation of the histogram analysis uses a thread pool. The performance of the calibration step is not improved with multiple CPU cores since POCL vectorizes array operations already for *cpu_01*.

Configuration *gpu_16* performs all OpenCL operations on the GPU. This heterogeneous approach combines CPU (e.g., I/O, histogram analysis) and GPU

(e.g., histogram creation, calibration). The calibration time is significantly reduced (4.86 s to 0.71 s). The GPU is much faster than the CPU for this task, which uses many trigonometric operations for the calibration of solar channels. Additionally, more parallelism can be used for the histogram generation and interpolation steps.

Loading data is now the most expensive step. Therefore, in configuration *gpu_16_io8*, multiple threads are used to load channels in parallel and reduce waiting on I/O. The resulting runtime (0.70 s instead of 4.35 s) is a significant improvement.

While a purely sequential execution allows processing a full disk scene in approximately 6 s, overlapping multiple scenes can reduce the average processing time, which is essential for time series. In configuration *analyze and load*, two scenes overlap to allow loading and analysis in parallel. This lowers the average runtime for a scene to 4.71 s. The configuration *complete* extends this strategy by overlapping all processing steps. This allows processing nine scenes concurrently. We improve the average time per scene to 4.21 s. Using CPU and GPU together provides the fastest setup, which also benefits from overlapping computation as both devices are constantly busy and waiting for work is minimized. Therefore, the heterogeneous processing approach has an additional benefit in this setup. However, the physical cores of the CPU are constantly fully utilized in this scenario. More CPU cores for histogram analysis and additional GPUs for the OpenCL tasks are necessary to reduce the runtime further. This strategy enables an average runtime similar to the longest-running step.

5.4.3 Use-Case: 10 Year FLS Climatology

The presented processing approach was applied to generate a ten-year FLS climatology for Europe. To our knowledge, it is the first one with spatial explicit FLS information and a 15 min resolution on a continental scale. As stated in the introduction of this chapter, this kind of dataset is essential for a deeper understanding of FLS formation and dissipation. It can also serve as a first step to create a ground fog product which could improve short-term forecasting of fog situations.

Figure 5.12 shows the resulting map of average FLS frequencies for November 2011. The spatial occurrence of FLS follows a northeast-southwest gradient, which is clearly visible. Additionally, hot-spots like the Po valley are visible.

We validate the processing implementation and the resulting FLS climatology against Meteorological Aviation Routine Weather Reports (METAR) stations

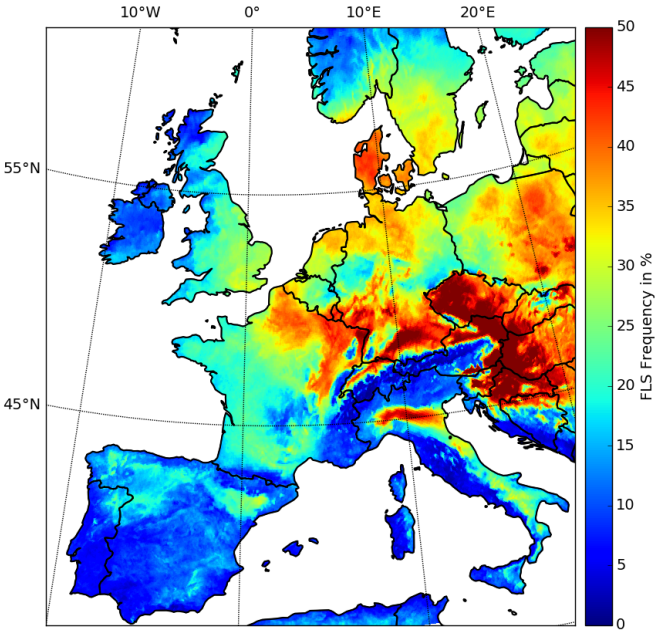


Figure 5.12: The resulting map of average FLS frequencies for November 2011.

[Egl+17]. The derived temporal and spatial variations of FLS were analyzed and show results matching with the ones of recent studies, see Egli *et al.* [Egl+17] for more details about results and validation.

5.5 Conclusion

In this chapter, we presented the design steps of a new processing tool for computing FLS masks for long time series of MSG SEVIRI data. Our approach merges and adapts existing schemes for FLS detection at day and night [CB07; CB08] to produce one homogeneous FLS mask. The FLS detection steps make use of tiles and overlapping windows for spatial processing. This enables results independent from the study areas size and allows shifting the study area location more easily.

Extending the modules for data access and preprocessing presented in Chapter 4, additional modules for FLS detection and refinement are developed. The modules for the FLS mask creation use efficient algorithms and parallel processing. Implementing raster processing with OpenCL allows intuitive single-pixel operations as well as complex operations like the histogram generation. We gain the best

performance when using both CPU and GPU together, and an additional speedup is achieved when all steps are run concurrently.

A high processing performance facilitates exploratory analyses and parameter sweeps. It is also required for new and future GEO satellites.

While we developed the processing tool with a focus on the ten-year FLS climatology for Europe, the implementation allows processing of other and larger areas as well. The implementation is open-source and published on GitHub[3]. Its modular design allows integrating additional tests as well as reusing modules for other tasks. The modules for data loading and pre-processing, as well as the FLS data, are already reused. Processing the entire MSG SEVIR time series in a short time also enables using novel machine learning approaches for different tasks like cloud or fog detection as well as rainfall estimation.

[3] `github.com/jdroenner/fls2`

6

Cloud Pixel Classification with Random Forest

This chapter focuses on the classification of cloud pixels in the MSG SEVIRI raster time series using Machine Learning (ML). As stated in the Introduction of this thesis (Chapter 1), clouds are essential for our climate and influence many aspects of life on earth. Remote sensing techniques using sensors on satellites provide the only way to generate global as well as long-term cloud datasets with spatial and temporal resolutions required for many applications [Stu+13].

Related work (Chapter 3.2) shows that, several approaches [Ban+09; TB11; TC13] have been developed to classify raster data produced from satellite sensors as clouded or cloud-free as well as to detect more complex cloud classes. They range from methods using a single threshold to more complex classification methods with multiple rules and dynamic threshold estimation to decide whether a pixel contains clouds or if it is cloud-free. One example is the Cloud Mask (CMa) from the CLAAS-2 dataset [Ben+17], which is introduced in Chapter 2.4.2. The technical document describing the single steps of the CMa algorithm [NWC13] contains over 32 pages with rules for different aspects of cloud detection. Therefore, a massive amount of expert knowledge is required to develop algorithms, which must also handle edge cases and complex effects caused by viewing geometry or the diurnal cycle of the sun.

A different approach to classifying remote sensing data is the employment of methods from the ML domain (see Chapter 3.3). This way, how a pixel is classified is learned, and the requirement for expert knowledge is shifted to data preparation and feature engineering as well as model training and evaluation. A multitude of methods based on Random Forest (RF) [ETB18], and Neural Networks (NN) [MDN17], are used for different classification tasks. Some detect single or multiple classes, while others solve regression problems.

RF [Bre01] is an ensemble learning method combining a large number of standard decision trees, as introduced in Chapter 2.5.1. Recently, RF has been shown to

perform very well for classifying multispectral satellite data from MSG SEVIRI [ETB18; Mey+16]. Therefore, this chapter focuses on RF models for cloud classification on MSG data. The developed models serve as a reference for an innovative method based on a CNN architecture for image segmentation, which is presented in the following chapter.

The different classification approaches using RF vary in the data preparation steps, channel combinations, or spatial statistics. As presented in Chapter 5, spatial information is significant for FLS classification. Since clouds are spatial entities, using only the information available at a single pixel's location, ignores spatial information. Therefore, spatial statistics about the closer environments (3×3 pixel or 5×5 pixel) are often incorporated, e.g., rodograms and two-dimensional variograms [ETB18]. To investigate the role of expert knowledge and spatial information for cloud pixel classification, two RF models are trained and evaluated: A naive model, which uses only the preprocessed MSG SEVIRI data, and a second model, which employs additional channel combinations and spatial context.

While the processing chain presented in Chapter 4, enables access to a large raster time series for training and evaluation, the availability of ground truth data is a major challenge when training a model on satellite raster data. The combination of satellite and ground truth data often requires the assignment of ground stations to individual raster pixels in specific projections. Data from MSG is in the GEOS projection, which is not used by most data sources and is also dependent on the location of the SSP. Additionally, using station data limits the number of pixels available for training and evaluation. Therefore, the presented work uses the CLAAS-2 CMa [Ben+17] as reference data, to avoid the ground truth problem. This has the benefit that both datasets match perfectly in space and time, and for each MSG SEVIRI scene, a CMa is available.

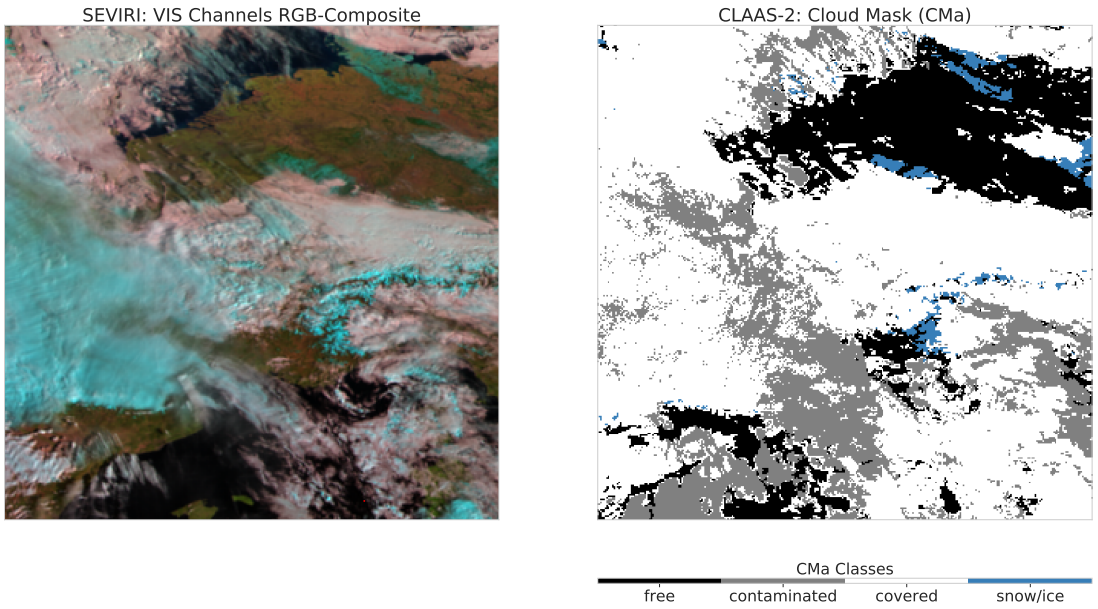
The structure of this chapter is as follows: First, we introduce the methods and data used to train the two RF models for cloud pixel classification. Section 6.2 evaluates the performance and quality of the different models. Finally, the results and the differences between the two models are discussed in Section 6.3.

6.1 Data and Methods

This section introduces the data and methods used for model creation, training, and evaluation. Similar to the previous chapters, the MSG SEVIRI data serves as

input data. To train and evaluate the RF models, we used the Scikit-learn library [Ped+11] for Python.

6.1.1 Study Area and Data



(a) (b)

Figure 6.1: An RGB-composite created from SEVIRI Channels 1–3 (a) and the corresponding CLAAS-2 CMa (b) for the study area used in this study. The displayed date is 22 February 2011 09:00:00 UTC.

The combination of MSG SEVIRI raster data and the CMa product from the CLAAS-2 dataset provides a vast amount of training and evaluation data. For practical reasons, we selected a square study area centered on Europe to train and evaluate the RF. Figure 6.1 shows the study area centered on Europe. It depicts, (a) an RGB-composite created from SEVIRI Channels 1–3 on the left- and (b) the CMa at the right-hand side.

Study Area

The study area is a square of 508×508 pixels centered on Europe. We selected this setup for two reasons: First, a square study area allows a more easy comparison with other methods where processing is split into regular tiles, e.g., the CNN-based

segmentation method presented in Chapter 7. Second, the selected area enables the evaluation of diurnal and seasonal influences as well as land/sea effects while focusing on a single image tile. As shown in Figure 6.1, it covers parts of the Atlantic in the west and continental areas in the east. In the north, it covers parts of Great Britain and Norway as well as the North and Baltic Seas, and the Mediterranean Sea in the south. This area, therefore, includes land and sea as well as lowlands and mountains (the Alps and the Pyrenees), that are often covered by snow.

In the example scene, large and small cloud areas, as well as areas covered with snow (Northern Germany and the Alps), are evident. Above the North Sea, cloud fragments can be seen whose surroundings appear cloud-free in the RGB image. In fact, there are cloud-contaminated pixels, as indicated by the CMa.

MSG SEVIRI Data

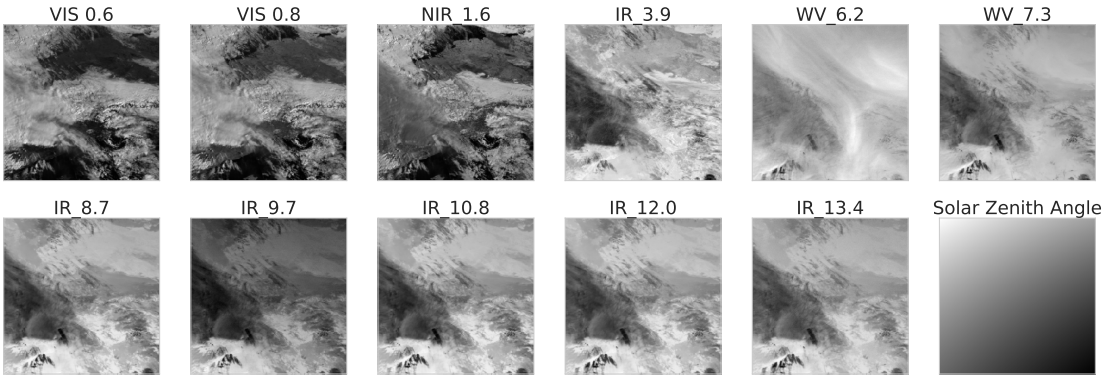


Figure 6.2: An overview of all SEVIRI channels from 22 February 2011 at 09:00:00 UTC. The channel names contain the sensitive spectrum or absorption band and the central wavelength in micrometers for each channel. Additionally, the bottom row on the right displays the SZA for the same date (black means small angle 0°, white large angle 90°).

The MSG SEVIRI data is presented in Chapter 2.4.1 and the required preprocessing in Chapter 4. Figure 6.2 presents an overview of the 11 main MSG SEVIRI channels on February 22, 2011, at 09:00:00 UTC for the selected study area. The channels are named as a combination of their spectrum or absorption band (VIS, NIR, WV, IR) and the central wavelength (0.6 µm − 13.4 µm). Additionally, the bottom row on the right displays the SZA for the same date (black means small angle 0°, white large angle 90°). For this chapter, the following aspects of the MSG SEVIRI raster time series are important:

1. MSG SEVIRI provides multispectral scenes with a temporal resolution of 15 min [Sch+02], resulting in 96 scenes per day and 35,040 scenes per year.
2. The channels 1–3 represent reflected irradiation (R_λ) and are therefore dark at night.
3. The channels 4–11 represent the emitted signal from the surface; therefore the signal is available at day and night. The values are represented as brightness temperatures (T_λ).
4. Channel 4 (IR_039) is influenced by solar radiation at daytime. Threshold-based methods use different rules for day and night.
5. The Solar Zenith Angle (SZA) determines the angle between a pixels zenith and the satellite. It correlates with the amount of solar radiation and therefore also the time of day.

CLAAS-2 Cloud Mask

Since it is not feasible to perfectly label all pixels of all MSG SEVIRI scenes manually, we use the well-validated Cloud Mask (CMa) from the CLAAS-2 dataset [Ben+17] as reference data. Chapter 2.4.2 contains a more detailed introduction of the CLAAS-2 CMa dataset. It provides different cloud properties derived from MSG SEVIRI data for the 12 years from 2004 to 2015. Each pixel of a MSG SEVIRI scene is classified as one of four classes: cloud-free, cloud-contaminated, cloud filed, or snow/ice. The CMa is, of course, a classification itself, but the validation results show that it has a very high quality.

The CMa algorithm includes several threshold tests as well as spatial tests, aggregating pixel neighborhoods that are applied to individual pixels. Pixels are first classified as cloud-contaminated using threshold values. If the value of a pixel is sufficiently distant from the threshold, it is classified as cloud-filled. Moreover, the tests distinguish between conditions caused by solar irradiation (day, night, and twilight) as well as land and sea [DGF13]. This is similar to the FLS detection method presented in Chapter 5. The separation allows to design rules for different surfaces and effects like sun-glint over the sea or the different properties of the 3.9 µm channel at day and night.

There are some known technical limitations that include classification of snow/ice-contaminated pixels. Since the corresponding rules in the CMa algorithm require reflectance information (Channel 1–3) to calculate the NDSI for snow classification, the snow/ice class is only available for pixels with solar irradiation and therefore only at daytime [DGF13].

Elevation Data

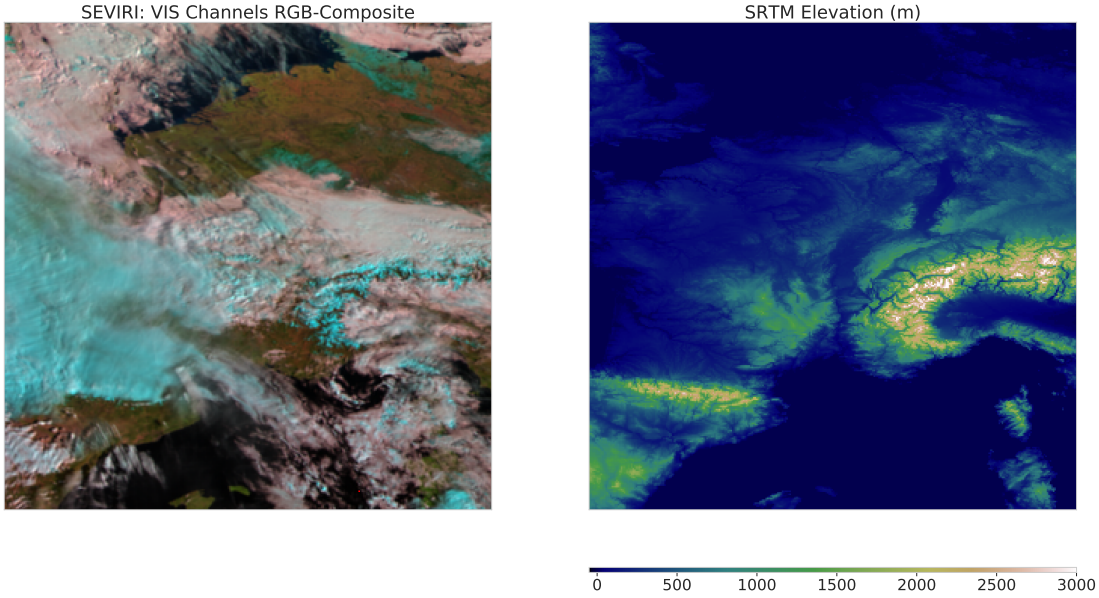


Figure 6.3: MSG RGB composite and the SRTM elevation data for the study area.

Elevation data is the base for the generation of a land/sea mask, and since orographic effects influence cloud creation and distribution, it is a potential relevant input variable. It is also commonly used as a variable for other studies using machine learning. One example is the investigation of the altitude of cloud bases by Egli *et al.* [ETB18]. The cloud base altitude is relevant for fog determination as its definition is a cloud touching the ground. We generated the elevation data for this study from the WorldClim dataset [FH17]. WorldClim provides global climate raster data with a pixel size of 1 km and contains a Digital Elevation Model (DEM) at the same resolution. The WorldClim DEM was derived from the SRTM DEM [Far+07] and provides the elevation above sea level in meters at each pixel's location. Since the MSG SEVIRI pixel size is 3 km×3 km at the SSP, the WorldClim DEM provides a matching resolution. Figure 6.3 shows the elevation data next to the RGB-composite of the example scene. It allows for easy discrimination between land and sea. Additionally, a correlation of higher elevations with snow-cover is evident for the Alps.

6.1.2 Model Setup and Training

RF classifies individual pixels and creates a classification of a raster by applying the model to each pixel. For model training and application, we use Scikit-learn [Ped+11], which uses the *Gini Impurity* as default error function. Both, RF and the Gini Impurity are explained in Chapter 2.5.1.

In an RF model, each tree is trained separately by taking a bootstrap sample from the training dataset. Each tree is trained by splitting a random subset of the original input features at each node. The data are split in such a way that within each subset, the Gini Impurity is minimal. This procedure is applied recursively until the training dataset has been processed.

Data preparation

We selected all MSG SEVIRI and CLAAS-2 data from the years 2004 to 2010, i.e., a total of six years, as the training data. Additionally, we select the data from 2012 as test and 2011 as validation data. The selection contains 30,040 scenes per year.

The loading and preprocessing modules, presented in Chapter 4, were used to prepare the MSG SEVIRI data. Other studies also use this preprocessing setup, e.g., Egli *et al.* [ETB18]. The MSG data is transformed into physical values and the full disk data is cropped to match the study area. The efficient loading and calibration modules allows the handling and transformation of eight years of SEVIRI data, totaling about 21 TB (compressed), in a short time.

The total training dataset included 205,000 scenes, and after removing all corrupted scenes, a total of 200,000 scenes remained. We remove a scene if either an MSG channel or the CMa is corrupt. Corruption is caused by glare effects or missing MSG channel segments. The remaining scenes are the input for the training of the naive model (Scenario RF1).

When used for multispectral classification, RF methods often use auxiliary data such as terrain elevation, the SVA, and handcrafted geo-statistical texture features [ETB18]. Therefore, we examine a second RF scenario (Scenario RF2) that incorporates additional features. For RF2, the input features are extended with multiple channel combinations, elevation, and geostatistical texture features. The channel differences are created based on expert knowledge and include the channel combinations also used and explained for the FLS detection in Chapter 5. One example is the difference of Channel 4 and Channel 9 (ΔT 3.9–10.8) which allows

the detection of FLS and cloud pixels. The geostatistical features include the variogram, the RODogram (ROD), and the Pseudo Cross Variogram (PCV). They are used and explained in more detail by Gloaguen *et al.* [GWM08] and Egli *et al.* [ETB18].

The **Variogram** γ_{Vario} represents the variation within a pixel's surrounding

$$\gamma_{Vario}(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} (v(x_i) - v(x_i + h))^2,$$

where $n(h)$ is the number of pairs for lag distance h. This is relevant for stratiform surfaces, e.g., FLS. Similar to Egli *et al.* [ETB18], we used $h = 1$ and a pixel surrounding of 5×5 pixels. For each pixel pair i the coordinates of pixels that are separated by a lag distance h are given by x_i and $x_i + h$. The raster value at locations x_i and $x_i + h$ are given by $v(x_i)$ and $v(x_i + h)$.

The **Rodogram** (ROD) γ_{ROD} takes the square root of all absolute differences

$$\gamma_{ROD}(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} \sqrt{|(x_i) - v(x_i + h)|}.$$

Additionally, statistics which include two raster bands are candidates for features used in the model training. One of them is the **Pseudo Cross Variogram** (PCV) γ_{PCV}, which calculates the differences of cross pairs between two data inputs $v(x_i)$ and $w(x_i + h)$

$$\gamma_{PCV}(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} (v(x_i) - w(x_i + h))^2.$$

Feature Selection and Parameter Tuning

To reduce the "curse of dimensional" [Bel15] a recursive feature selection was conducted. After the relevant features are selected, the model parameters are tuned, and the two RF models are trained.

Combining the multitude of possible features, including single channels, channel combinations, elevation, and spatial statistics, can cause the size of the dataset

to inflate, which increases the processing time. For this step, we follow the example of Egli *et al.* [ETB18]. To find the most useful information content from the input data and to reduce the noise caused by irrelevant or redundant data, we executed a recursive feature elimination. In this process, models are trained and the least important feature is removed after each step. The quality of a trained model is calculated using an out-of-bag R^2 score. The RF implementation provides the R^2 score, which indicates the global fit of a linear regression. The best model with the most relevant features was selected, as reported in Table 6.1.

Table 6.1: Final set of features used for RF training in the second model. MSG bands are denoted with their central wave length. PCV stands for Pseudo Cross Variogram and ROD stands for RODogram.

MSG Bands	**Band Differences**	**Texture Features**	**Additional Data**
VIS 0.6	ΔT 7.3–12.0	PCV (8.7,10.8)	Terrain Elevation
IR 8.7	ΔT 8.7–10.8	PCV (3.9,10.8)	Satellite Viewing Angle
IR 10.8	ΔT 10.8–12.0	PCV (10.8,12.0)	
IR 12.0	ΔT 3.9–7.3	ROD (8.7)	
	ΔT 3.9–10.8	ROD (10.8)	
		ROD (12.0)	

Table 6.1 shows the selection of channels and combinations. Similar features are also identified by Egli *et al.* [ETB18] for fog detection using RF. Most of the channels and channel combinations are also found in the set of rules for FLS classification, which are derived from expert knowledge (Chapter 5). The reflectance in the VIS channel with a wavelength at 0.6 µm obviously allows the identification of bright areas, which are snow or clouds at daytime. The rules used for FLS detection contain channels and combinations selected by the feature selection (see Chapter 5): The IR channels can serve as a proxy for the cloud top altitude, and the channel at 10.8 µm is used to detect ice clouds. The channel combination ΔT 3.9–10.8 is used as the main indicator to identify FLS and cloud pixels using a threshold. Other examples are ΔT 8.7–10.8 as well as ΔT 10.8–12.0, which are used to identify cirrus clouds. The PCVs for these channel combinations are also selected. Additionally, the RODs for the IR channels are presented. They can indicate either a stratiform surface or convective cells.

Based on the method used by Egli *et al.* [ETB18], the total number of trees and the number of features used to split each node were selected. This is an iterative process. For a range of 50 to 300 trees and a range from 1 to 10 features, the

model performance was evaluated. A setup of 150 trees and 5 features per split was found to provide the best performance.

Model Training

We use the Scikit-learn package for Python [Ped+11] to train the two RF models. RF1 uses only the 11 MSG SEVIRI channels as inputs, while RF2 uses the features shown in Table 6.1.

For training, a pre-sampling step reduced the number of feature-arrays. A feature-array represents all features at a pixel's location, including channel values and combinations, as well as elevation, and spatial statistics. Remember that the training data includes 200,000 scenes from the years 2004–2010. With 508×508 pixels per scene the sum of all pixels and therefore feature-arrays is 51,612,800,000 pixel. A random selection of 100 scenes per month and 1000 pixels per scene was selected. Therefore, the sum of pixels (and feature-arrays) is reduced to a total of 7,200,000. This approach reduces the amount of data inserted into the model for training and allows to fit the data into the main memory. However, this is simply an optimization step since the RF training also takes samples from the training data. Experiments with larger training sets validated that the performance of the model does not change.

The following code snippet shows how RF1 is trained with Scikit-learn. RF2 uses the same code for training with different independent variables.

Listing 6.1: Training Random Forest in Python

```
from sklearn import preprocessing
from sklearn.ensemble.forest import RandomForestClassifier
import numpy as np

data = load_data() # placeholder for data access

# define independent variables and target variable
independent_vars = ["VIS006", "VIS008", "IR_016", "IR_039", "
    WV_062", "WV_073", "IR_087", "IR_097", "IR_108", "IR_120", "
    IR_134"]
target_var = "cloud_mask"

# split the 3D-input data into training and target arrays
training_input  = np.asarray(data[independent_vars])
training_target = np.asarray(data[target_var])

# initialize a scaler and normalize the data
scaler = preprocessing.StandardScaler().fit(training_input)
```

```
17 training_input_transformed = scaler.transform(training_input)
18
19 # initialize and train the RF model
20 clf = RandomForestClassifier(n_estimators=150, max_features=5)
21 clf = clf.fit(training_input_transformed, training_target)
```

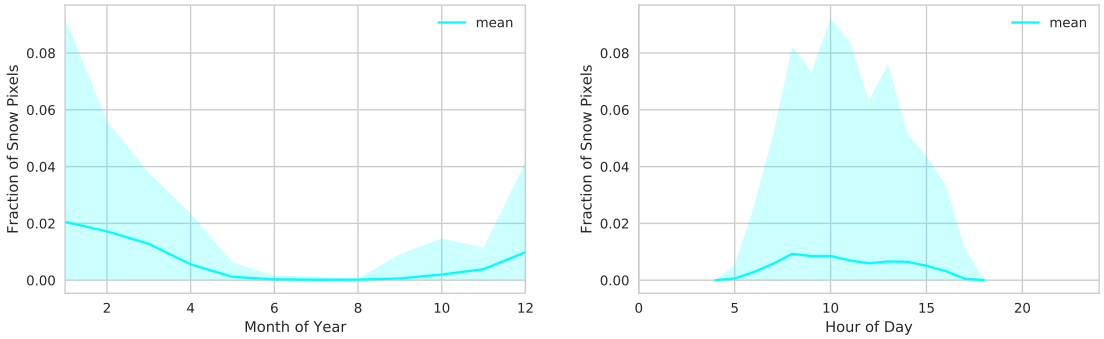
In Line 1 and 2, the Scikit-learn [Ped+11] modules for preprocessing and RF are imported. Additionally, the Numpy library is imported for n-dimensional array handling in Line 3. A placeholder method for data loading is used in Line 5. The data is assumed to contain all variables for RF1 and RF2. Therefore the required variables are defined in Lines 8 and 9. The data is split into the feature arrays *training_input* and the reference value *training_target* in Line 12 and 13. To train the models, the data requires normalization. The *StandardScaler* in Line 16 calculates the standard deviation and the mean values of the features. These values are then used in Line 17 to scale the values to unit values $]0, 1[$. Finally, we initialize the *RandomForestClassifier* in Line 20 with 150 trees (*estimators*) and five features to use for spliting. This example shows that RF is straightforward to use and can handle vast numbers of input features with a sampling approach.

6.2 Evaluation

This section reports the experimental results of the two different model setups RF1 and RF2. First, we introduce the data and the metrics used for evaluation. Then, we show an example classification for a single scene using both methods, which also indicates the time required for the classification. Finally, the evaluation is concluded by comparing the global statistics of both setups. We investigate the robustness of the different models by comparing the distribution and variance of the classifications of all individual scenes. Additionally, we investigate the impact of seasonality and the time of day on the models.

6.2.1 Evaluation Data

To evaluate and compare the performance of the models, we randomly selected 8000 scenes from the year 2011. The training data includes only scenes for the years 2004–2010, which implies total independence of the evaluation data and the training data. After removing corrupted scenes, we used a total of 7977 scenes,



(a) (b)

Figure 6.4: Distribution of the fraction of snow pixels per CMa scene as a function of months (a) and hour of day (b).

i.e., 23.4% of the 35,040 scenes created per year, for evaluation and comparison.

One of the challenges for cloud detection is the treatment of snow. While cloud and cloud-free pixels are equally distributed over scenes, the snow class is limited to daytime in the CLAAS-2 CMa scenes. Additionally, the occurrence of snow pixels in the study area is tied to seasonality. Figure 6.4 shows the variance as well as the mean snow fraction per scene aggregated by month of the year (a) and hour of the day (b). High fractions of snow pixels only occur during the winter months on the northern hemisphere where the maximum fraction per scene is 9.2%. In the summer months, the maximum and the mean fraction of snow pixels per scene fall below 0.05%. The right plot shows the dependency on sunlight for snow pixel information in the CMa product. Note that snow pixels are not available for night hours. During the day, snow pixels mostly occur between 8 a.m. and 1 p.m., i.e. when there is solar irradiation. Therefore, the correct detection of snow in the MSG SEVIRI data is challenging, and the reference data does not represent the real truth. However, the model might correctly learn to identify snow at night, but the comparison with the reference data will rate this as a misclassification.

6.2.2 Evaluation Metrics

The evaluation metrics are calculated based on the comparison of the CMa data with the results of the trained model scenarios. We use the evaluation metrics introduced in Chapter 2.5.2. For each scene of the evaluation data, the classification

generated by a model is compared pixel by pixel with the CLAAS-2 CMa. The results are represented as confusion matrices for each class as well as combined (overall). A confusion matrix contains four counters to track the true positive classified pixels (correctly predicted events), the false positive (incorrectly predicted events), the true negative (correctly predicted non-events) and the false negative ones (incorrectly predicted no-events). We create local confusion matrices for each scene independently to compare individual scenes and analyze the distribution of the evaluation metrics over time. Additionally, the local confusion matrices are combined into global confusion matrices that cover all scenes and represent 837,393,552 classified pixels in total. This is the foundation to calculate the following indicators for the performance of each model. The following metrics are used to evaluate the performance of each model: accuracy, Probability of Detection (POD), Probability of False Detection (POFD), False Alarm Ratio (FAR), and the Heidke Skill Score (HSS). See Chapter 2.5.2 for more details.

6.2.3 Example Scene and Performance

The scene introduced in Section 6.1 was also selected to present a first example of the RF classifications. Figure 6.5 shows the CMa as classified by RF1 and RF2 on the left and the right, respectively. Remember, the CLAAS-2 CMa of this scene and the RGB composite is also included in this chapter and depicted in Figure 6.1. We visualize the four classes (cloud-free, cloud-contaminated, cloud-filled, and snow/ice) and add a different colorization for pixels where the class selected by the model does not match the original class in the CMa. By looking at both results, one can already reasonably conclude that RF2 provides results closer to the original. The results also show that the classification of cloud areas, in general, is correct. However, both RF models generate larger contiguous misclassified areas in the northeast.

Both models, RF1 and RF2, were trained and executed on a machine with Intel i7-5930K CPU (3.5GHz), 64 GB RAM, and an NVidia GeForce GTX TITAN X GPU. Classifying an MSG SEVIRI scene cropped to the study area requires the application of the RF models to all pixels individually. The implementation uses the Scikit-learn library and runs on the CPU. We used all 12 CPU cores for RF training and application. The classification of the scene requires data loading, normalization, and application of the trained model. The RF2 scenario requires the generation of spatial features and channel combinations for each scene, which takes 1.44 s. While both setups require a similar time to classify all pixels, the RF2 model has an average runtime of 1.86 s per scene, which is clearly

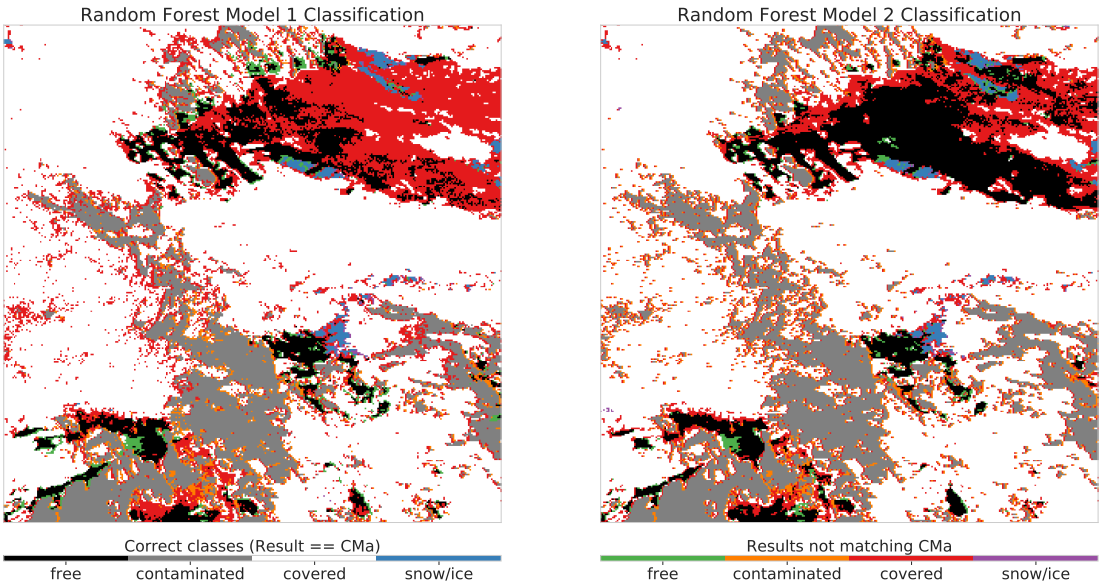


Figure 6.5: Random forest classifications of RF1 and RF2 for the example scene (22 February 2011 09:00:00 UTC). Pixels matching the CMa reference data are colored in the CMa color scheme. The classes of pixels not matching the CMa are indicated with colors.

dominated by the generation of the channel combinations and the spatial statistics.

The following code snippet shows how the Scikit-learn library is used to classify a selected MSG SEVIRI scene using the RF1 model without additional features:

Listing 6.2: Applying a Random Forest in Python

```
from sklearn import preprocessing
from sklearn.ensemble.forest import RandomForestClassifier
import numpy as np
import pickle as pkl

scaler = pkl.load("msg_cma_scaler.pkl")
model = pkl.load("msg_cma_model.pkl")

data = load_msg_scene() # placeholder for data access
data_2d_shape = (508, 508) # or (324, 324)

# define independent variables and target variable
independent_vars = ["VIS006", "VIS008", "IR_016", "IR_039", "
    WV_062", "WV_073", "IR_087", "IR_097", "IR_108", "IR_120", "
    IR_134"]
```

```
14
15 # select the input features
16 input_data  = np.asarray(data[independent_vars])
17
18 # normalize the data
19 input_transformed = scaler.transform(training_input)
20
21 # apply the RF model
22 cloudMask_rfResult = model.predict(testing_input_transformed)
23
24 # reshape result into 2d-array
25 cloudMask_rfResult = cloudMask_rfResult.reshape(2d_shape)
```

In Line 1 and 2, we import the Scikit-learn modules for preprocessing and RF. Additionally, the Numpy library is imported for n-dimensional array handling in Line 3, and the serialization library Pickle is loaded in Line 4. The Pickle library is used to de-serialize the data scaler and the trained model. Both were created in Listing 6.2. A placeholder method for data MSG SEVIRI scene loading is used in Line 9. The data is assumed to contain the SEVIRI channel values for RF1 and additionally, the statistics for RF2. Therefore, the required variables are defined in Line 13. The data is refined into *input_data* in line 16. In this step, feature arrays represent a slice through all features for each individual pixel, e.g., an array with all the channel values for RF1. To apply the model, the data is normalized first. The *StandardScaler* initialized and fitted in the training step is used in Line 19 to scale the values to unit values (0.0 to 1.0). Finally, the trained *RandomForestClassifier*, which is loaded in Line 7, is used to predict the CMa classes for the input data in Line 22. As the last step, the data is re-shaped into a two-dimensional array matching the MSG SEVIRI scene in Line 26.

Table 6.2: Statistics for the scene from 22 February 2011 09:00:00 UTC.

(a) RF Scenario 1: 11 channels

Class	Accuracy	HSS	POD	FAR	POFD	Bias
Combined	0.887	0.813				
1: cloud-free	0.915	0.828	0.910	0.103	0.081	1.013
2: cloud-cont.	0.954	0.801	0.779	0.103	0.017	0.882
3: cloud-fill	0.899	0.796	0.899	0.125	0.101	1.024
4: snow/ice	0.999	0.764	0.656	0.060	0.000	0.716

(b) RF Scenario 2: 11 channels, differences, spatial features and elevation.

Class	Accuracy	HSS	POD	FAR	POFD	Bias
Combined	0.931	0.885				
1: cloud-free	0.937	0.865	0.928	0.083	0.062	1.012
2: cloud-cont.	0.984	0.934	0.928	0.039	0.006	0.967
3: cloud-fill	0.942	0.882	0.934	0.064	0.052	0.998
4: snow/ice	0.999	0.668	0.631	0.256	0.0004	0.888

Table 6.2(a),(b) represents the results for the example scene (22 February 2011 09:00:00 UTC). The impression, that RF2 generates more accurate results is confirmed by the statistics in Table 6.2(a),(b), where the overall accuracy is 0.887 for RF1 while the RF2 model archives an accuracy of 0.931. The HSS represents the skill gained by the model in comparison to a random classification. For the model using only the channel data, the HSS over all classes is 0.813 while the model with additional information has an overall skill of 0.885. Both RF classifications show problems at the north-eastern corner, where large cloud-free areas are classified as clouded. However, the RF1 model misclassifies a much larger area. This area in northern Germany has bright, probably sandy, ground pixels. Additionally, in the northwest, misclassified pixels correspond to the coastline. Both are common problems [SK88; SMM04] that can occur when classifying clouds. The statistics agree with that observation and show a cloud-free HSS of 0.828 for RF1 and 0.865 for RF2. The FAR and the POFD for cloud-filled pixels are also much higher for RF1 than for RF2. In the same area, both RF models underestimate snow patches, which correlates with the HSS for snow classification. It is 0.764 for RF1, while RF2 has a skill of 0.668.

6.2.4 Global Results

To evaluate the performance and to compare the trained RF models, the global statistics of all classes are presented in Table 6.4(a)–(e). For this part of the evaluation, the results from all selected test scenes were combined to generate global confusion matrices.

Table 6.4: Statistics for 7977 test scenes from 2011.

(a) Metrics for the **cloud-free** class

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
RF1	0.918	0.830	0.901	0.100	0.070	**1.001**
RF2	**0.937**	**0.868**	**0.920**	**0.077**	**0.052**	0.997

(b) Metrics for the **cloud-contaminated** class.

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
RF1	0.953	0.838	0.847	0.109	0.024	0.956
RF2	**0.986**	**0.951**	**0.960**	**0.042**	**0.009**	**1.002**

(c) Metrics for the **cloud-filled** class.

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
RF1	0.903	0.804	0.898	0.119	0.093	1.018
RF2	**0.945**	**0.889**	**0.939**	**0.064**	**0.050**	**1.003**

(d) Metrics for the **snow/ice** class.

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
RF1	**0.9989**	**0.791**	**0.678**	**0.035**	**0.0001**	0.713
RF2	0.9985	0.712	0.636	0.149	0.0004	**0.785**

(e) Metrics for all classes **combined**.

Scenario	Accuracy	HSS
RF1	0.890	0.824
RF2	**0.934**	**0.895**

Table 6.4(a) shows the results for the cloud-free class. The model RF2, trained with the features from the feature selection dominates all metrics except bias. While

the accuracy for both models is above 0.9, the HSS is 0.83 for RF1 and 0.868 for RF2. The POD for cloud-free pixels is lower for RF1 (0.901) than for RF2 (0.92). This correlates with the impression from the example that both models overestimate the cloud classes. It is also visible when comparing FAR and POFD for both models.

The metrics for the cloud-contaminated class are shown in Table 6.4(b). RF2 shows the best metrics for this class. In contrast to the cloud-free class, RF2 also shows the smallest bias. Again, both accuracy values show high accuracy values (> 0.95). However, the HSS differs significantly. RF1 only shows an HSS of 0.838 for the cloud-contaminated class while the RF2 HSS is 0.951.

Table 6.4(c) shows the metrics for the cloud-filled class. For this class, the findings are similar to the previous classes. RF1 clearly shows the lower accuracy and skill (HSS).

The snow class results are presented in Table 6.4(d). Here, the accuracy values are very high for both models. This is related to the small fraction of snow pixels per scene, as indicated by Figure 6.4. For this class, RF1 shows better results then RF1. The HSS of RF1 is 0.791 while RF2 is near 70% with 0.712. The POD values are below 70% for both models, and the FAR for RF2 is significantly higher than for RF1. The POFD is very low for both models.

The accuracy and the HSS for all classes combined are presented in Table 6.4(e). For both metrics, model RF2 shows the highest values, with a clear difference to RF1.

6.2.5 Robustness

For a robust model, values such as accuracy and skill (HSS) should have a high median value and a small range for each class over individual scenes. Additionally, different classes should also be at a similar accuracy and skill levels to keep the overall skill level even if distributions of pixels per class changes. Figure 6.6 shows the box-plots of HSS and POD for both trained RF models in the same order as the tables in the previous section. Here, we create the box-plots from the individual results of each of the 7977 evaluation scenes. The box represents the second and third quartile, containing 50% of the scenes, while the median is represented as the (green) dividing line.

For a robust model, i.e., a model that produces reliable results and a constant quality level, we expect small boxes. This means that the HSS values of a class are, in the optimal case, not changing between scenes with a different class distribution.

Additionally, the whiskers, representing the overall data distribution, should cover only a small area so that there is no extreme difference between high and low values.

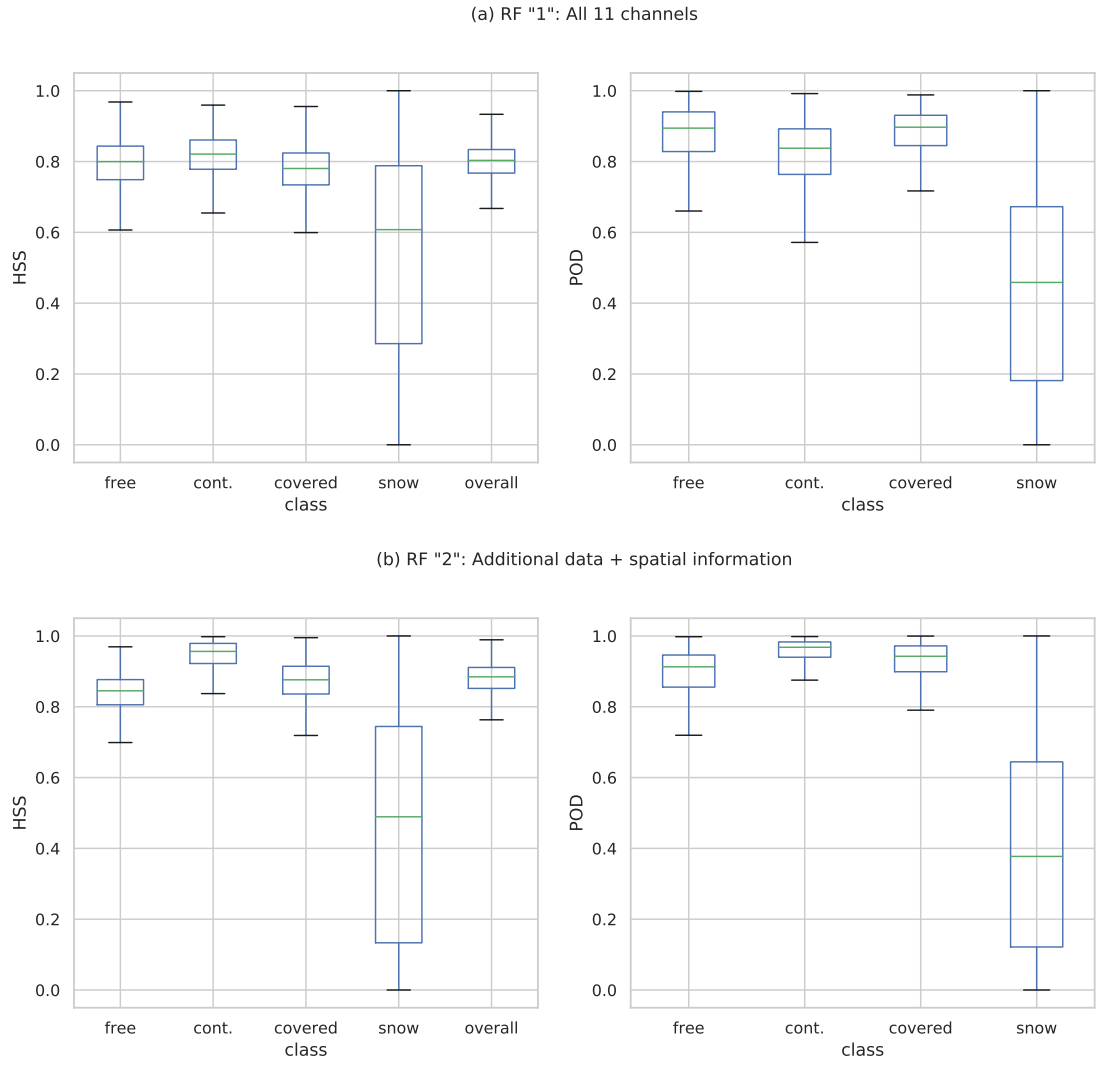


Figure 6.6: Box-plots of HSS and POD generated from all results of 7977 test scenes. HSS for all scene pixels combined is also included.

The first row in Figure 6.6 represents the first RF model (RF1) and the second row RF2. The left plots show the HSS, and the right plots show the POD of the models. We generate all results from the selected set of 7977 scenes.

The RF1 model shows HSS median values that are centered around 0.8 for the

cloud-free and cloud classes. The second and third quartiles cover ranges between 0.6 and 0.97. The snow class shows an HSS median value just above 0.6, and the second and third quartiles cover a range from 0.3 to 0.8 while the absolute range is from 0.0 to 1.0. For all classes combined the median is 0.8 and the second and third quartiles cover a range from 0.85 to 0.95.

The RF2 model, which uses MSG SEVIRI channels, elevation data, and spatial statistics, improves all metrics except for snow when compared with RF1. It is visible that the HSS median and the second and third quartile box for most classes show value ranges at a higher level. Noticeable is the cloud-contaminated class that shows a very high median value and a small box between 0.9 and 1.0. Compared to the previous RF model, the HSS for snow is lower. This is also indicated by the POD plot.

RF1 has more stable median values if the cloud-free and the cloud classes are compared. However, the values are at higher levels for RF2. This is also true for all classes combined. The snow class is the only exception where RF1 shows better skill and POF than RF2. However, the median of both models is low.

6.2.6 Seasonal and Diurnal Dependencies

The snow class in the CMa data depends on seasonality since there is less snow in Europe in the summer months. Additionally, the CMa algorithm uses the reflectance information from solar channels to classify pixels as snow-covered. Therefore, there are no snow pixels in the CMa data at night [DGF13]. The influence of this aspect requires further investigation. Figure 6.7 shows the HSS values grouped and aggregated by the month of the year on the left side and by the hour of the day on the right side.

For the RF1 model, the seasonal variation of the cloud classes and the combined classes is quite low. All classes show values between 0.7 and 0.9, which are very constant over the year. However, the snow/ice class reaches an HSS level over 0.8 during winter months but drops below 0.4 during the summer months. A dependency on the hour of the day is also clearly visible in the right plot. In the night hours, no HSS is shown for snow/ice pixels while it rapidly rises (and falls) with solar irradiation between 5 a.m. and 6 p.m. to a maximum of 0.7. For the seasonal and the diurnal visualization, the HSS of the cloud-contaminated class shows an interesting pattern. At night and during winter, the HSS of this class is higher than the other classes, and at daytime and during the summer months it drops below the HSS of the other classes.

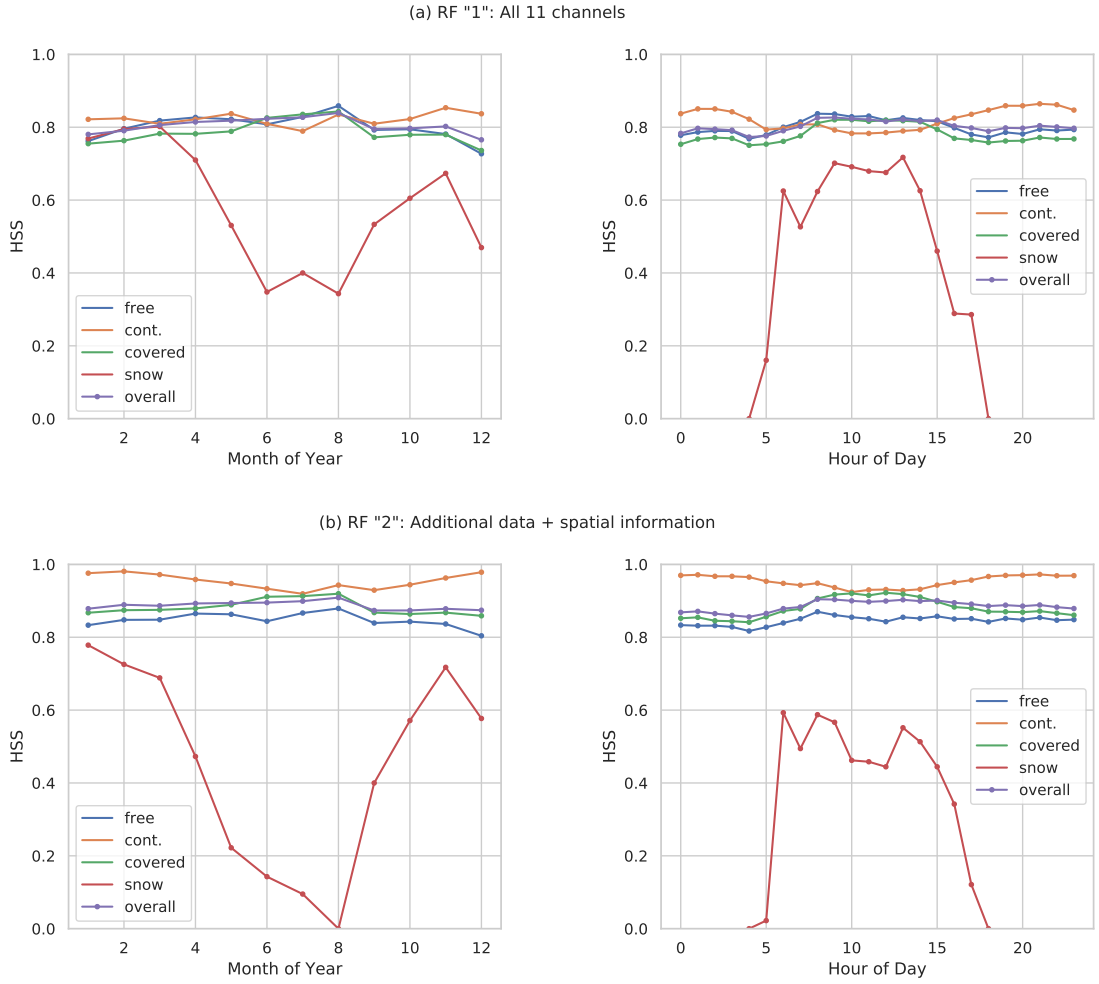


Figure 6.7: Plots of HSS median values from all results of 7977 test scenes grouped by month of the year and hour of the day.

The patterns for the RF2 model show significant seasonal and diurnal dependencies, while still resembling the ones shown by the other model. This one shows clearly visible different values for all classes for daytime and month of the year. Cloud classes stay between an HSS of 0.8 and 1.0. However, they show more deviating value levels. The snow/ice class again shows different patterns. While the performance in the winter months is similar to the other RF model, the performance drops to a value near 0.0 during summer. The diurnal distribution shows similar patterns, but snow/ice stays below an HSS of 0.6 during daytime.

6.3 Discussion and Conclusion

As discussed in the introduction and Chapter 3, threshold-based, and machine learning approaches require expert knowledge to identify relevant rules and features. Productive cloud masks such as CLAAS-2 CMa use many rules to handle different characteristics of multispectral data where various parameters such as the position of the sun play a role in the classification. The RF used in our study learns a classification of cloud pixels from the multispectral SEVIRI data. However, as found in Section 6.2, it is not sufficient to use only the MSG SEVIRI data. By adding additional data, handcrafted spatial information, and channel combinations, the RF2 model shows better metrics. Although the feature selection for the model has led to improvements for the cloud classes, it considerably worsened the results of the snow class. Therefore, the application of traditional learning methods such as RF requires considerable expert knowledge in terms of handcrafted features and feature selection.

The importance of the spatial information introduced by the geostatistical features is also visible when looking at the plotted results of the individual scenes. The plots show a higher HSS for RF2. An interesting exception is the HSS of the snow class, which is lower in summer for RF2.

The third issue is the need to support nowcasting and processing of large time series concerning low latency and fast processing, respectively. The classification using RF is very fast. However, the generation of the geo-statistics and channel combination dominates the time to classify a single scene, which takes 1.86 s for the RF2 model.

While the RF method is easy to apply, an extensive amount of expert knowledge is required to identify and create features with spatial information, and meaningfully channel combinations. Additionally, the sampling approach of RF allows training on large time series, but not all pixels are considered for learning. The next chapter presents a novel method for cloud pixel classification, where the spatial information is included automatically. Therefore, the demand for expert knowledge and time is investigated, as well as the processing time.

7

Cloud Segmentation with Convolutional Neural Networks

The previous chapters present different aspects and applications of the Meteosat Second Generation (MSG) raster time series. Chapter 4 comprises the preprocessing steps necessary to access and transform the raw data in an efficient way. This data is then used to classify pixels as covered by Fog and Low Stratus (FLS) in Chapter 5. Chapter 6 contains a Machine Learning (ML) method to classify the pixels as free or cloud-covered. Both presented classification approaches, as well as many others [Ban+09; TB11; TC13], have been developed for the classification of single pixels using rules, which are based on expert-knowledge (see Chapter 3.2). The available methods range from using a single threshold to complex sets of rules to decide whether a pixel is clouded or cloud-free. Related work (Chapter 3.3) also shows that methods from the machine learning domain are used increasingly to classify multiple and complex classes.

We see three major drawbacks of current methods. First, most of them focus on the independent classification of individual pixels and rarely use spatial information that extends beyond adjacent pixels. Thus, these pixel-based approaches ignore the fact that clouds are a spatially continuous and highly dynamic phenomenon. However, spatial structures and dynamics contain essential information, e.g., the surface structure of clouds is particularly meaningful. Therefore, spatial statistics about the closer environment of individual pixel are often used. This approach is also employed in Chapter 6 and for FLS detection in Chapter 5. Nevertheless, these statistics represent only a small part of the available spatial information, and they do not take into account the spatial relationships and dynamics in broader areas and scales. Second, domain experts must manually create and select the most appropriate features for the classification task, similar to the classification using Random Forest (RF) presented in Chapter 6. This is a quite cumbersome and time-consuming task due to a vast number of possible features. Since a larger number of training data is beneficial to increase the accuracy of current machine learning techniques, it is useful to employ entire datasets for training. However, it

is not feasible to create all possible features manually for datasets in the terabyte range with many channels. At the same time, sampling and feature selection also reduces the potential training data. Third, products derived from satellite data are either required in a timely manner for nowcasting or in the form of large time series for analysis. To react as quickly as possible to a situation, the processing time must be significantly shorter than the production rate of the corresponding satellite. Additionally, fast processing is particularly necessary for processing large time series.

To address these problems, we propose to use a deep learning approach that has shown great promises in a broad spectrum of applications [LBH15]. In particular, Convolutional Neural Networks (CNN) [KSH12; LSD15; RPB15; He+17] are highly suitable for object detection, image classification, and segmentation. Unlike other methods, CNNs automatically learn the most important features without involving domain experts to create or select them manually. Despite their advantages, related work (Chapter 3.3) shows that existing methods for cloud classification follow the approach to individually treat a pixel when a model is trained and later when a trained model is applied. In contrast, we use a new version of a CNN architecture for image segmentation to classify an entire image holistically. This fully exploits the spatial information during model generation and enables the efficient evaluation of the trained model since it avoids redundant operations on adjacent pixels. While in previous work the segmentation capabilities of CNNs were used for objects such as houses in spatial RGB images [Xu+18], a novel contribution of our work is the extension of a CNN architecture for multispectral geostationary satellite data covering the continuous nature of clouds in the atmosphere. This is a challenging task because multispectral data consist of multiple channels with different characteristics that can change between scenes for reasons such as the diurnal cycle of the sun as well as changing atmospheric constitution and temperature.

To summarize, our contributions are as follows. We present a novel cloud classification method using a CNN architecture for image segmentation. All pixels are classified simultaneously instead of individually. We show that the proposed CNN architecture is able to handle multispectral satellite data and highly dynamic clouds. Our approach shows excellent results concerning classification quality and runtime performance.

7.1 Data and Methods

This section presents the architecture of the proposed Cloud Segmentation CNN (CS-CNN) in detail. CS-CNN is designed to create high-quality spatial classifications from multispectral remote sensing data. Additionally, the presented CS-CNN relies only on the original data, in this case, the MSG SEVIRI channel raster data. Therefore, it does not require handcrafted features to incorporate spatial information. CS-CNN is based on a CNN architecture for image segmentation and promises very fast processing times by using GPUs and avoiding redundant operations.

The standard application of CNNs is to classify the content of gray-scale or RGB images, as presented in Chapter 2.5.2. A CNN architecture for such a task is usually designed as a concatenation of convolution, pooling, and fully-connected neural layers [LBH15]. This leads to a single result vector with probabilities for each trained class, e.g., the probability of cat content in an image. The proposed CS-CNN aims to derive cloud masks from multispectral remote sensing data. Therefore, the used architecture needs to ingest multispectral satellite scenes and generate a segmentation where each pixel of the input scene is assigned to a specific cloud class. Since there is no pre-built architecture or a pre-trained model for this task, we developed the CS-CNN.

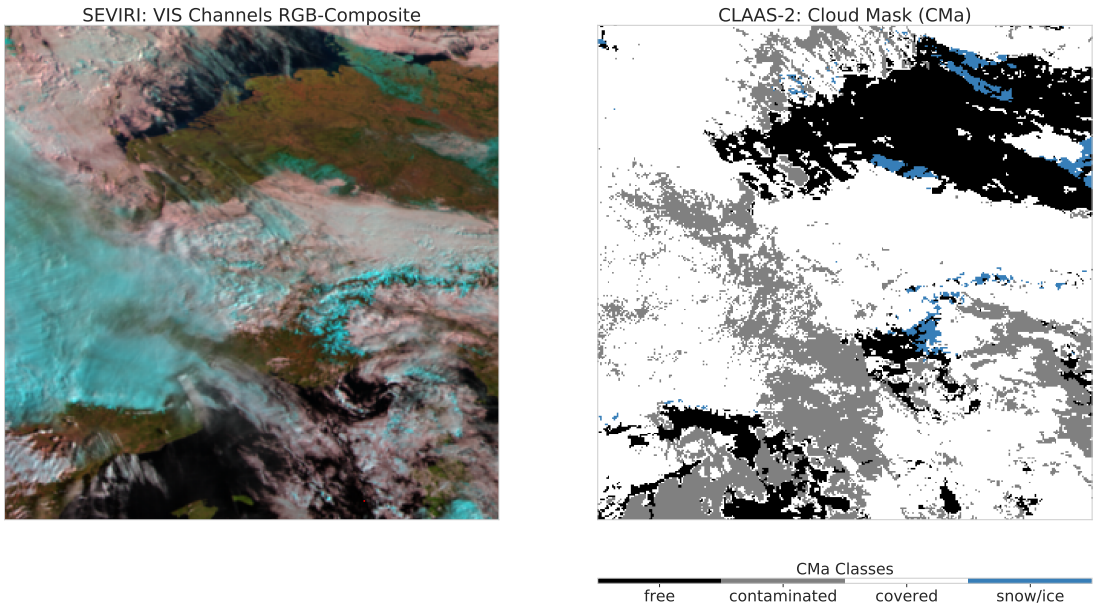
Training such a CNN is a supervised learning process that requires a correctly labeled classification for each training input. During training, CS-CNN iterates over each training scene and loads all input channels together with the correct classification information (reference data). CS-CNN classifies the scene and computes the classification error or loss based on the provided ground truth. Backpropagation is then used to gradually adapt the weights of all layers of CS-CNN to minimize the loss. In our approach, the output grid also contains each pixel of the input grid.

Training and evaluation of the CS-CNN uses the same data and study area as the RF approach in Chapter 6. The MSG SEVIRI raster time series (Chapter 2.4.1) provides multispectral scenes with a temporal resolution of 15 min [Sch+02], resulting in 96 scenes per day and 35,040 scenes per year. It is available since the start of the MSG program in 2004. Therefore, the training, as well as the evaluation, can be conducted with many data. Since it is not possible to perfectly label all these scenes manually, the well-validated Cloud Mask (CMa) from the CLAAS-2 dataset [Ben+17] is used as reference data in our work. This dataset is presented in more detail in Chapter 2.4.2. It is, of course, a classification itself, but validation results show that it has a high quality. Additionally, noise in the training data is often used to prevent overfitting of the trained model [GBC16].

Furthermore, for other segmentation tasks, generated reference data are beneficial since they increased the number of training data [Alo+17]. Reusing the setup from Chapter 6 enables a comparison with a proven state-of-the-art ML method. In this case RF.

Before we explain the architecture of the CS-CNN in depth, we revisit the datasets and the connected challenges.

7.1.1 MSG SEVIRI and CLAAS-2 CMa Data



(a) (b)

Figure 7.1: An RGB-composite created from SEVIRI Channels 1–3 (a) and the corresponding CLAAS-2 cloud mask (b) for the study area used in this study. The displayed date is February, 22, 2011 at 09:00:00 UTC.

The training, testing, and evaluation data used for the CS-CNN is identical to the previous chapter (see Chapter 6.1 for more details). Figure 7.1 (a) shows the RGB-composite created from SEVIRI Channels 1–3 on the left and the CMa (b) on the right. This is the same scene as described in Chapter 6. In the example, large and small cloud areas, as well as areas covered with snow (Northern Germany and the Alps), are evident. The combination of both datasets presents a perfect opportunity to evaluate the capabilities of CNNs with a segmentation

architecture on a massive number of data for training, validation, and evaluation.

We selected a square sub-region of the MSG SEVIRI full-disc (see Chapter 2.4.1) centered on Europe as our study area. CS-CNN takes multiple raster layer with a size of 508×508 pixels as input. Therefore, the selected study area was setup to match this size. It is visualized in Figure 7.1(a) next to the CLAAS-2 CMa. The focus on Europe, enables the evaluation of the influence of diurnal and seasonal as well as land/sea effects.

The reference data CLAAS-2 CMa provides cloud information for the years from 2004 to 2015. Each pixel of a MSG SEVIRI scene is classified into one of four cloud classes: cloud-free, cloud-contaminated, cloud-filled, or snow/ice [DGF13] (see Chapter 2.4.2).

MSG SEVIRI and CLAAS-2 data pose challenges for training and evaluation. Four MSG SEVIRI channels are sensitive to reflected solar irradiation. The VIS-channels are dark at night and Channel 4 covers both solar reflection and thermal emission during the day and only thermal emission during the night. Therefore, the CNN has to cope with the diurnal cycle and variable feature availability.

CLAAS-2 has known technical limitations that include the classification of snow or ice contaminated pixels. Since the corresponding rules in the CMa algorithm require information from the solar Channels 1–3 to classify snow, the snow/ice class is only available for pixels with solar irradiation and therefore only at daytime [DGF13]. At night, snow-covered pixels are classified as cloud-free.

7.1.2 Cloud Segmentation CNN (CS-CNN) Architecture

Due to its remarkable performance in biomedical image segmentation, we decided to use the U-Net architecture [RPB15] as the technical platform for our Cloud Segmentation CNN (CS-CNN). The architecture of the CNN is composed of 9 blocks for down- and up-sampling, as illustrated in Figure 7.2. Its almost symmetrical architecture resembles several other CNN architectures for image segmentation [LSD15; NHH15; BKC17] in terms of the composition of a sequence of down-sampling layers, followed by another sequence of up-sampling layers. We used the Caffe deep learning framework [Jia+14] to implement CS-CNN as a graph of connected convolution layers.

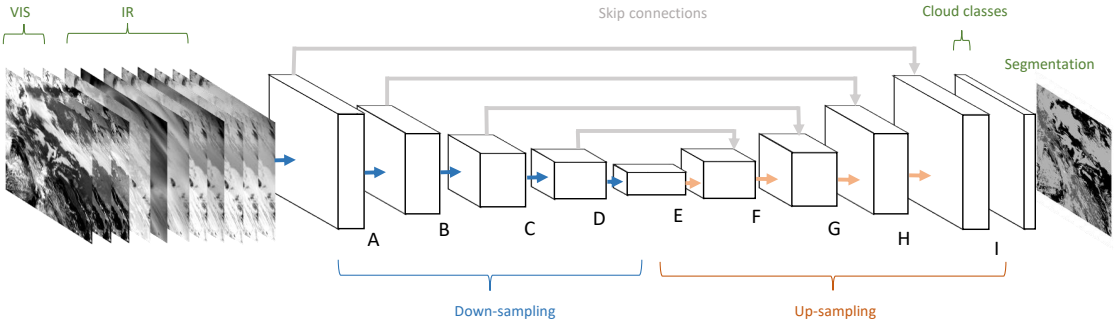


Figure 7.2: CS-CNN architecture used for cloud segmentation.

The left-hand side of Figure 7.2 shows the input for the network. The spatial coverage is 508 × 508 pixels, which matches the size of the selected study area. The depth of the input depends on the number of selected channels per scene. To evaluate the influence of the different channel characteristics, either seven or eight infrared channels (IR) and optionally the solar reflectance Channels 1–3 are used. The sequence of learning blocks, consisting of multiple convolution layers, is shown next to the input layer. Each block has a label (A–I) attached that refers to Table 7.1 where the details of the convolution layer blocks and their connections are given. The right-hand side in Figure 7.2 depicts the computed cloud classes that return, for each pixel, the probabilities per class. Note that the output of the CS-CNN is a symmetric subset of the input with a size of only 324 × 324 pixels since CS-CNN uses unpadded convolutions. For each of these pixels, the network outputs a five-dimensional probability vector. It represents the degree of membership for each class. The meaning of the image right of the cloud classes differs when the CS-CNN is in the learning phase or the application phase. In the learning phase, the image illustrates the reference data. As introduced, this is the CLAAS-2 CMa, which is used calculate the error/loss of a CS-CNN result for each training iteration. This information is then backpropagated through CS-CNN from right to left to adapt the weights of the different convolutional layers in each block. In the application phase, the image represents the result of a trained model. In this case, CS-CNN assigns the class with the highest probability of each pixel as a result.

Table 7.1: Architecture for cloud segmentation.

Block	Layer Type	Output Size	Skip Connection
input	input	$\{7, 8, 11\} \times 508 \times 508$	
A	3×3 conv, relu	$32 \times 506 \times 506$	
A	3×3 conv, relu	$32 \times 504 \times 504$	I
A	3×3 conv, stride 2	$32 \times 252 \times 252$	
B	3×3 conv, relu	$64 \times 250 \times 250$	
B	3×3 conv, relu	$64 \times 248 \times 248$	H
B	3×3 conv, stride 2	$64 \times 124 \times 124$	
C	3×3 conv, relu	$128 \times 122 \times 122$	
C	3×3 conv, relu	$128 \times 120 \times 120$	G
C	3×3 conv, stride 2	$128 \times 60 \times 60$	
D	3×3 conv, relu	$256 \times 58 \times 58$	
D	3×3 conv, relu, dropout	$256 \times 56 \times 56$	F
D	3×3 conv, stride 2	$256 \times 28 \times 28$	
E	3×3 conv, relu	$512 \times 26 \times 26$	
E	3×3 conv, relu, dropout	$512 \times 24 \times 24$	
E	3×3 deconv, relu	$256 \times 48 \times 48$	
F	3×3 conv, relu	$256 \times 46 \times 46$	
F	3×3 conv, relu	$256 \times 44 \times 44$	
F	3×3 deconv, relu	$128 \times 88 \times 88$	
G	3×3 conv, relu	$128 \times 86 \times 86$	
G	3×3 conv, relu	$128 \times 84 \times 84$	
G	3×3 deconv, relu	$64 \times 168 \times 168$	
H	3×3 conv, relu	$64 \times 166 \times 166$	
H	3×3 conv, relu	$64 \times 164 \times 164$	
H	3×3 deconv, relu	$32 \times 328 \times 328$	
I	3×3 conv, relu	$32 \times 326 \times 326$	
I	3×3 conv, relu	$32 \times 324 \times 324$	
output	3×3 deconv, relu	$5 \times 324 \times 324$	

CS-CNN has similar principles as U-Net but differs in some aspects. The architecture consists of blocks of convolutional layers that are visible in Figure 7.2. Table 7.1 lists the composition of each block. In contrast to U-Net, the number of filters in each layer is much lower in CS-CNN to obtain a lighter network. This reduces the required calculations and improves the classification runtimes. Each block in CS-CNN consists of two layers performing 3×3 pixel convolutions with the rectified linear unit (ReLU) as activation function. In Blocks D and

E, dropout is used to prevent over-fitting. Down-sampling happens as the last operation in Blocks A–D. In contrast to the U-Net, we performe down-sampling by strided convolutional layers instead of pooling layers. Strided convolutions are common convolution operations, but they use a larger pixel stride. This enables the CNN to learn a specific down-sampling operation for each layer. To generate a segmented image, the down-sampling sequence is followed by up-sampling. The last layers in Blocks E–I in Figure 7.2 perform up-sampling by transposed convolutions (deconv). Moreover, CS-CNN also utilizes features from down-sampling layers again in the corresponding up-sampling counterparts. These so-called skip connections have become an indispensable component in a variety of deep neural architectures.

A further difference between CS-CNN and U-Net is that CS-CNN does not require data augmentation, such as elastic deformations, to increase the amount of training data. Augmentations are beneficial when the training dataset is small. This is, however, not necessary in our cloud segmentation study, because many annotated raw satellite images are available by combining the SEVIRI and CLAAS-2 data. To handle these large datasets on commodity hardware, efficient preprocessing and a batch-wise training approach are required.

7.2 Model Training and Experimental Setup

In this section, we present the experimental setup, and the methods to measure the performance of CS-CNN. The selected evaluation data and the used metrics are identical to these in Chapter 6.2. CS-CNN was trained on three different input channel configurations that were selected to evaluate the influence of channels sensitive to solar irradiation. To compare CS-CNN against a classifier for individual pixels, we additionally compare CS-CNN with a competitive method. The two RF models that are described in detail in Chapter 6 are used for this purpose.

7.2.1 Model Training

CS-CNN training was performed for three scenarios that contain decreasing levels of solar influenced channels. This setup was selected to evaluate the necessity and the impact of solar channels on the segmentation. All data from MSG SEVIRI and the CLAAS-2 CMa generated between the years 2004 and 2010, i.e., a total of six years, were selected as the training data for all models. Since MSG and CMa are

generated in 15-min intervals, this results in approximately 205,000 scenes. After removing all corrupted data, a total of 200,000 scenes remained for training. A scene was removed if either an MSG channel or the CMa was corrupt. The remaining scenes were randomly inserted in a list that was used for training. Identical to the RF models presented in Chapter 6, the year 2011 was selected for evaluation and the year 2012 for testing the model while training.

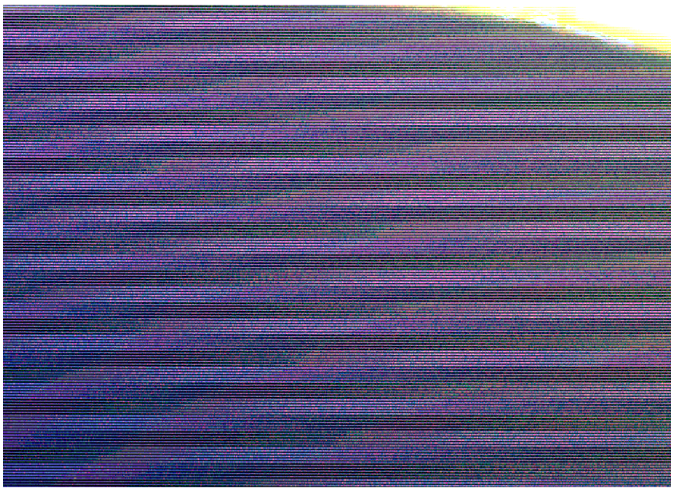


Figure 7.3: Two possible errors in the MSG SEVIRI data. Under certain sun angles, e.g., at sunrise, glare effects may occur. One of these is located in the upper right corner of the image, where the image is entirely white. The rest of the image shows scan line artifacts.

Scenario A contains all 11 channels including Channels 1–3 that are dark at night. Recall that each training iteration consists of one scene with all 11 channels and the corresponding CLAAS-2 cloud mask representing the "truth". While CS-CNN can handle dark areas without changes, glare effects in the reflectance Channels 1–3 can disturb the training process. We found that spiking loss (or error) values occurred during training when specific night scenes were ingested. This is because glare effects occur at night when the affected pixel should be dark. For certain sun angles, light either directly reaches the sensor or is reflected by the surface, while the other pixels of the scene are dark. For the study area covering Europe, this is especially true in the summer months. To solve this issue, we added a preprocessing step that masks a channel as entirely dark when a glare effect occurs. Another issue is caused by scan-line artifacts, which arise especially at night. Figure 7.3

shows examples of both effects. A glare effect is located at the top right, while scan-line artifacts are visible on the left side.

In contrast to Scenario A, the second scenario (B) removes the VIS channels which contain reflected solar irradiance. It includes the thermal Channels 5–11 and Channel 4 that is dominated by the solar signal during daytime and the thermal signal at night. This reduces the influence of changing characteristics caused by the transition between day and night that CS-CNN needs to handle. Note that reducing the number of input channels does not create other changes in CS-CNN.

In Scenario C, we use only Channels 5–11 that are not directly influenced by solar irradiation for training. Thus, the channel characteristics will not change between day and night. Therefore, this scenario is expected to be the easiest to train.

The training of all three scenarios was performed using Caffe on a PC with Intel i7-5930K CPU (3.5GHz), 64 GB RAM, and four Nvidia GeForce GTX TITAN X GPUs. All MSG SEVIRI scenes, as well as the CMa data, were preprocessed and cropped to match the size of the model input layer. We stored the processed ready-to-use data as compressed files. This minimizes not only I/O operations but also removes redundant operations if more epochs (iterations of the training data) are used for training. We stored the compressed data of approximately 4TB on a Toshiba X300 High-Performance HDD.

We selected the Adaptive Moment Estimation (ADAM) [KB14] as our optimization method since it provides excellent results and fast convergence. This is beneficial, since training one epoch, which contains 200,000 scenes, takes approximately 34 h. To validate the models during and after training, we used a random sample of 1200 scenes from 2012. Note that this test dataset is segmented repeatedly after a small number of iterations and a larger number of scenes increases the required time for model training.

Figure 7.4 shows the training loss and the accuracy measured while training all model scenarios using the selected test dataset. The loss of all three scenarios rapidly drops until it is below 0.3, then it keeps decreasing with an increasing number of training iterations. At the same time, the accuracy of each scenario keeps increasing. After 30,000 iterations, which is approximately the number of scenes produced per year, the accuracy of Scenarios B and C are near 90%, while Scenario A is already above 90%. Overall, Scenario A shows the highest accuracy and the lowest loss, but only with a small margin. The training was stopped after 200,000 iterations as further training does not change the accuracy, which stays constant after 125,000 iterations. This effect is similar to the training of the RF

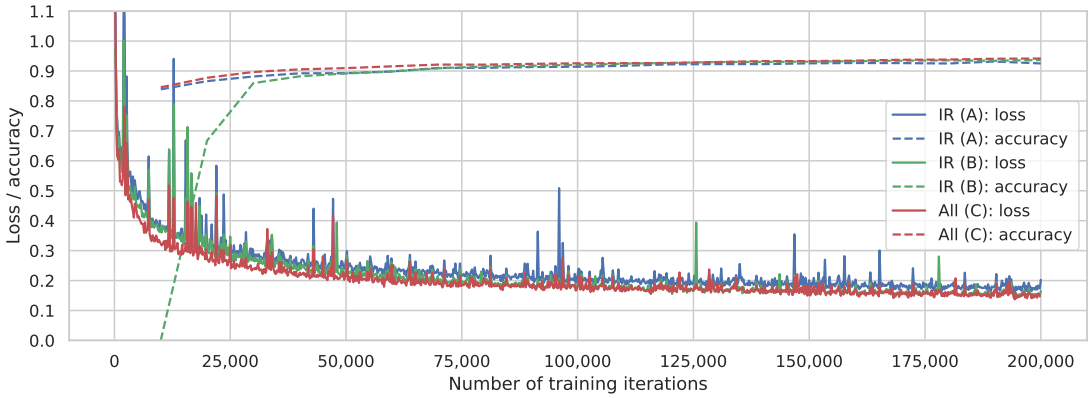


Figure 7.4: Training loss and accuracy for all three model variations.

model presented in Chapter 6, where a larger set of training data does not increase the model's performance.

7.2.2 Evaluation Setup

To evaluate and compare the performance of the models, we randomly selected 8000 scenes from 2011. This is the same dataset used for the RF evaluation in the previous chapter. The training data includes only scenes from the years 2004–2010, which implies total independence of the evaluation data and the training data. After removing corrupted scenes, we used a total of 7977 scenes, i.e., 23.4% of the 35,040 scenes created per year for evaluation and comparison.

One of the challenges for cloud detection is the treatment of snow, as presented in Chapter 6. While cloud and cloud-free pixels are equally distributed over scenes, the snow class in the CLAAS-2 CMa is limited to daytime. Additionally, the occurrence of snow pixels in the study area is tied to seasonality. High fractions of snow pixels only occur during the winter months on the northern hemisphere where the maximum fraction per scene is 9.2%. Additionally, snow pixels are not available for night hours. During the day, snow pixels mostly occur between 8 a.m. and 1 p.m., i.e., when there is much sunlight.

Competitive Method: Random Forest

We compare the CS-CNN to the RF models presented in Chapter 6. RF classifies individual pixels and creates a classification by applying the model to each pixel. Recently, RF has been shown to perform very well for classifying multispectral

satellite data [ETB18; Mey+16]. The essentials of RF are described in Chapter 2.5.1.

Equivalent to CS-CNN training, the same set and order of training scenes were used the two RF scenarios. RF1 is referenced as Scenario D. It uses all 11 SEVIRI channels as independent input features. However, when used for multispectral classification, RF methods often use auxiliary data such as terrain elevation, the SVA, and handcrafted geo-statistical texture features [ETB18]. Therefore, we created the second RF scenario (Scenario E), where we introduce additional features. These are combined channels (differences) as well as geo-statistical texture features. Note that these geostatistical features provide information about spatial structure in the SEVIRI data. Afterwards, a recursive feature elimination was conducted to select the most relevant features. The selected features are reported in Chapter 6 in Table 6.1.

Evaluation Metrics

We use the evaluation metrics introduced in Chapter 2.5.2. For each scene of the evaluation data, the classification generated by a model is compared pixel by pixel with the CLAAS-2 cloud mask. The results are represented as confusion matrices for each class as well as combined (overall). A confusion matrix contains four counters to track the true positive classified pixels (correctly predicted events), the false positive (incorrectly predicted events), the true negative (correctly predicted no-events) and the false negative ones (incorrectly predicted no-events). For each scene independently, local confusion matrices are generated to compare individual scenes and analyze the distribution of the evaluation metrics over time. Additionally, the local confusion matrices are combined into global confusion matrices that cover all scenes and represent 837,393,552 classified pixels in total. Equally to Chapter 6, this is the foundation to calculate the following indicators for the performance of each model. First, the accuracy of a class is the portion of positive and negative pixels classified correctly. Second, the POD returns the portion of pixels that correctly belong to a class. Third, the POFD gives the relative amount of pixels falsely classified. Fourth, FAR provides the probability of a false classification if a pixel is classified as belonging to a class. Finally, the HSS is calculated to evaluate the skill of each model. The metrics are described in detail by Jolliffe and Stephenson [JS03] and are also used by similar studies (e.g., [ETB18; Beu+18; Mey+16]).

7.3 Evaluation Results

The goal of this section is to report our experimental results of the different scenarios introduced in Sections 7.2.1 and 7.2.2. First, we show an example classification for a single scene using both classification approaches, the CS-CNN, and the RF method. Then, we compare the global statistics of all three deep learning scenarios (Scenarios A–C) and the two RF scenarios (Scenarios D and E). We investigate the robustness of the different models by comparing the distribution and variance of the classifications of all individual scenes. Additionally, we investigate the impact of seasonality and the time of day on the models. Finally, we show the influence of pixels located close to the spatial boundaries of cloud entities.

7.3.1 Example Scene and Performance

The scene introduced in Section 7.1 was also selected to present a first example. Figure 7.5 shows the cloud mask of CS-CNN and RF on the left-hand side and right-hand side, respectively. The RGB composite of the SEVIRI Channels 1–3 as well as the CLAAS-2 cloud mask of this scene are depicted in Figure 7.1. Again, we visualize the four classes (cloud-free, cloud-contaminated, cloud-filled, and snow/ice). Additionally, we added a different colorization for pixels where the class selected by the model does not match the original class in the CMa.

We calculate confusion matrices by comparing the model results with the original cloud mask pixel by pixel. Table 7.2(a),(b) represent the result for the example scene (22 February 2011 09:00:00 UTC). By looking at both results, one can already reasonably conclude that CS-CNN provides results close to the original. The RF results also show that cloud areas are correctly classified in general. However, larger contiguous misclassified areas are recognizable in the northeast. This is confirmed by the statistics in Table 7.2(a),(b), where the overall skill is 0.905 for the CS-CNN while the RF archives a skill of 0.885. While both results deviate slightly from the original, the CS-CNN shows differences to the original model only at spatial boundaries of entities. The RF classification shows many differences at the north-eastern corner, where a large cloud-free area is classified as clouded. These areas in northern Germany are bright, probably sandy, ground pixels and in the northwest pixels corresponding to the coastline. Both are common problems [SK88; SMM04] that occur when classifying clouds.

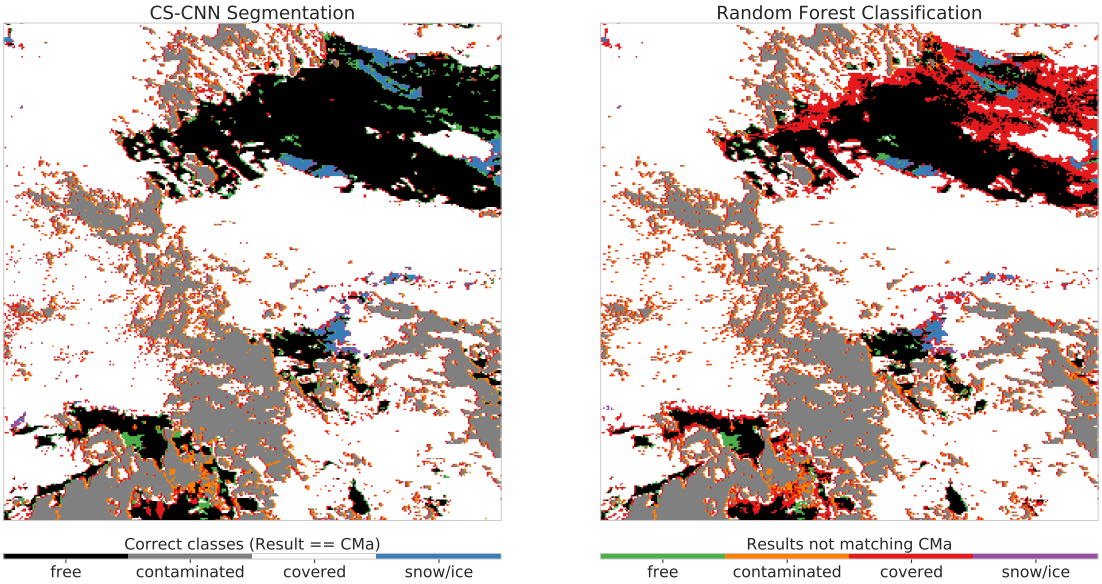


Figure 7.5: CS-CNN and random forest classification for the example scene (22 February 2011 09:00:00 UTC). Pixels matching the CMa reference data are colored in the CMa color scheme. The classes of pixels not matching the cloud mask are indicated with colors.

Table 7.2: Statistics for the scene from 22 February 2011 09:00:00 UTC.

(a) CS-CNN Scenario A: 11 channels

Class	Accuracy	HSS	POD	FAR	POFD	Bias
Combined	0.942	0.905				
1: cloud-free	0.961	0.920	0.957	0.047	0.036	1.004
2: cloud-cont.	0.973	0.885	0.886	0.082	0.013	0.968
3: cloud-fill	0.950	0.898	0.946	0.060	0.047	1.006
4: snow/ice	0.999	0.824	0.788	0.124	0.0002	0.913

(b) RF Scenario E: 11 channels, differences, spatial features and elevation.

Class	Accuracy	HSS	POD	FAR	POFD	Bias
Combined	0.931	0.885				
1: cloud-free	0.937	0.865	0.928	0.083	0.062	1.012
2: cloud-cont.	0.984	0.934	0.928	0.039	0.006	0.967
3: cloud-fill	0.942	0.882	0.934	0.064	0.052	0.998
4: snow/ice	0.999	0.668	0.631	0.256	0.0004	0.888

The statistics agree with that observation and show a cloud-free skill of 0.920 for the CS-CNN and 0.865 for the RF model. In the same area, the RF model also underestimates snow patches, which correlates with the skill for snow classification. It is 0.668, while the CS-CNN has a skill of 0.824. The confusion matrices generated for both models show that undetected snow pixels are equally classified by CS-CNN as cloud-free (257 of 595) and cloud-filled (314 of 595), while RF assigns most of them as cloud-filled (754 of 945).

Both models were run on a machine with Intel i7-5930K CPU (3.5GHz), 64 GB RAM, and an Nvidia GeForce GTX TITAN X GPU. Data loading is implemented equally for all methods. We exclude the loading time from the execution times stated in the following. Caffe can run CS-CNN (A) on CPU and GPU. The execution time on the CPU for CS-CNN is 1.24 s while a run on the GPU takes only 25 ms on average.

The application of the RF model (E) to all pixels of a scene takes 1.86 s on average. The implementation uses the Scikit-learn library and runs on the CPU. It uses all 12 CPU cores for RF training and application. The RF scenario requires the generation of spatial features and channel combinations for each scene, which takes 1.44 s and is included in the runtime. While GPU based RF implementations are also available (e.g., [Sch+15]), the Scikit-learn results allow a comparison of the accuracy of the technologies. Additionally, the overhead for feature generation and channel combinations is independent of the RF implementation.

7.3.2 Global Results

To evaluate the performance and to compare the trained CS-CNN models with the RF models, the global statistics of all classes are presented in Table 7.4(a)–(e). For this part of the evaluation, the results from all scenes were combined to generate global confusion matrices. Therefore, the presented tables show the results calculated using all pixels considered in the evaluation.

Table 7.4(a) shows the results for the cloud-free class. CS-CNN Scenario A, trained with all channels, dominates all metrics except POD and bias, while Scenario C, trained without solar channels, shows the highest POD, it has higher FAR and POFD values. Scenario D, the RF trained with all channels, shows the smallest bias, but the other values are not as good as in the CS-CNN scenarios. The accuracy and HSS values of RF Scenario E are also smaller than the CS-CNN values.

Table 7.4: Statistics for 7977 test scenes from 2011.

(a) Metrics for the **cloud-free** class

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
A	**0.960**	**0.918**	0.949	**0.047**	**0.032**	0.996
B	0.953	0.902	0.958	0.074	0.051	1.032
C	0.943	0.884	**0.959**	0.098	0.067	1.057
D	0.918	0.830	0.901	0.100	0.070	**1.001**
E	0.937	0.868	0.920	0.077	0.052	0.997

(b) Metrics for the **cloud-contaminated** class.

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
A	0.972	0.903	0.917	0.078	0.016	0.996
B	0.972	0.902	0.907	0.067	0.014	0.975
C	0.969	0.891	0.883	0.059	0.013	0.942
D	0.953	0.838	0.847	0.109	0.024	0.956
E	**0.986**	**0.951**	**0.960**	**0.042**	**0.009**	**1.002**

(c) Metrics for the **cloud-filled** class.

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
A	**0.948**	**0.895**	**0.943**	0.062	0.048	1.005
B	0.943	0.884	0.924	**0.055**	**0.042**	0.979
C	0.934	0.865	0.908	0.060	0.047	0.968
D	0.903	0.804	0.898	0.119	0.093	1.018
E	0.945	0.889	0.939	0.064	0.050	**1.003**

(d) Metrics for the **snow/ice** class.

Scenario	Accuracy	HSS	POD	FAR	POFD	Bias
A	**0.9993**	**0.882**	**0.860**	0.089	0.0003	**0.950**
B	0.9985	0.753	0.820	0.356	0.0010	1.176
C	0.9981	0.698	0.779	0.449	0.0013	1.227
D	0.9989	0.791	0.678	**0.035**	**0.0001**	0.713
E	0.9985	0.712	0.636	0.149	0.0004	0.785

(e) Metrics for all classes **combined**.

Scenario	Accuracy	HSS
A	**0.941**	**0.906**
B	0.935	0.896
C	0.924	0.878
D	0.890	0.824
E	0.934	0.895

The metrics for the cloud-contaminated class are shown in Table 7.4(b). RF Scenario E shows the best values for this class. This scenario is also the only scenario that overestimates this class, as indicated by the bias. Additionally, the other RF scenario (D) shows values clearly below the values of the CS-CNN scenarios. The CS-CNN metrics show the best accuracy and skill values for all channels (A), and the lowest error values are achieved for the scenario without solar influenced channels (C).

Table 7.4(c) shows the metrics for the cloud-filled class. Here, CS-CNN Scenario A again shows the highest accuracy, skill, and POD values. CS-CNN trained without the reflectance channels (Channels 1–3) shows the lowest error values. Both RF scenarios (D and E) have higher error values than any CS-CNN scenario. Scenario D clearly shows the lowest accuracy and skill; Scenario E shows the second best.

The snow class results are presented in Table 7.4(d). Here, the accuracy values are very high for all scenarios, which is related to the small fraction of snow pixels per scene, as indicated by Figure 6.4. The HSS of CS-CNN Scenario A (0.882) is the only one exceeding 80%. The POD is also clearly dominated by the CS-CNN scenarios, while both RF scenarios are below 70%. The FAR values are high for CS-CNN without solar channels and lowest for RF scenarios, while the POFD is very low for all scenarios.

The accuracy and the HSS for all classes combined are presented in Table 7.4(e). For both metrics, CS-CNN Scenario A shows the highest values. The scenario with the second-best values is CS-CNN Scenario B with a minimal difference separating it from RF Scenario E. While the CS-CNN without solar channels is very close to the other ones, the RF Scenario D shows the lowest values with a clear difference to the next best scenario.

7.3.3 Robustness

For a robust model, values such as accuracy and skill (HSS) should have a high median value and a small range for each class over individual scenes. Additionally, different classes should also be at a similar accuracy and skill levels to keep the overall skill level even if distributions of pixels per class changes. Figure 7.6 shows the box-plots of HSS and POD for all five models in the same order as the tables in the previous section. Here, we create box-plots from the individual results of each of the 7977 evaluation scenes. The box represents the second and third quartile, containing 50% of the scenes, while the (green) dividing line represents the median. For a robust model, i.e., a model that produces reliable results and a

constant quality level, we expect small boxes. This means that the HSS values of a class are, in the optimal case, not changing between scenes with different class distribution. Additionally, the whiskers, representing the overall data distribution, should cover only a small area so that there is no extreme difference between high and low values.

The first row in Figure 7.6 represents CS-CNN Scenario A. The left plot shows the HSS; the right plot shows the POD generated from all 7977 scenes. For HSS, almost identical median values and value ranges are displayed for the cloud-free and cloud classes as well as for all classes combined. The snow class shows an HSS median value of 0.8, and the second and third quartiles cover a range from 0.5 to 0.9.

CS-CNN Scenario B is shown in Figure 7.6(b), indicating the impact of removing Channels 1–3. Most classes show a pattern similar to Scenario A, with slightly lower values. Similar to the global results in the previous section, the values of the ice/snow ice-class show the most significant changes. The value ranges of HSS and POD shift down and grow. Most notably, the HSS medium value for snow/ice is now 0.6.

CS-CNN Scenario C, represented by Figure 7.6(c), shows patterns similar to the previous two scenarios. Cloud and cloud-free classes have slightly lower values compared to Scenario B, which is also true for all classes combined. Again, the ice/snow class deviates significantly from the other classes. Here, the median of the HSS is 0.3, while the POD shows a median similar to Scenario B.

Comparing the RF Scenario D, which uses all SEVIRI channels, with the equivalent CS-CNN Scenario A, it can be seen that the overall HSS and the overall distribution are considerably lower and cover larger ranges. The median for the cloud classes and overall show a uniform level near 0.8, while the HSS for ice/snow is at 0.6. The POD for all CS-CNN scenarios has a nearly identical pattern, which can also be seen for this RF model, albeit with significantly lower values and larger value ranges.

RF Scenario E, which uses all SEVIRI channels, elevation data, and spatial statistics, improves all metrics except for snow when compared with the previous RF model. When compared with the CS-CNN Scenario A, it is visible that the HSS median and the second and third quartile box for most classes show value ranges at a lower level. The exception is the cloud-contaminated class that shows a very high median value and a small box between 0.9 and 1.0. Compared to the previous RF model, the HSS values for snow/ice are lower, which the POD plot also represents.

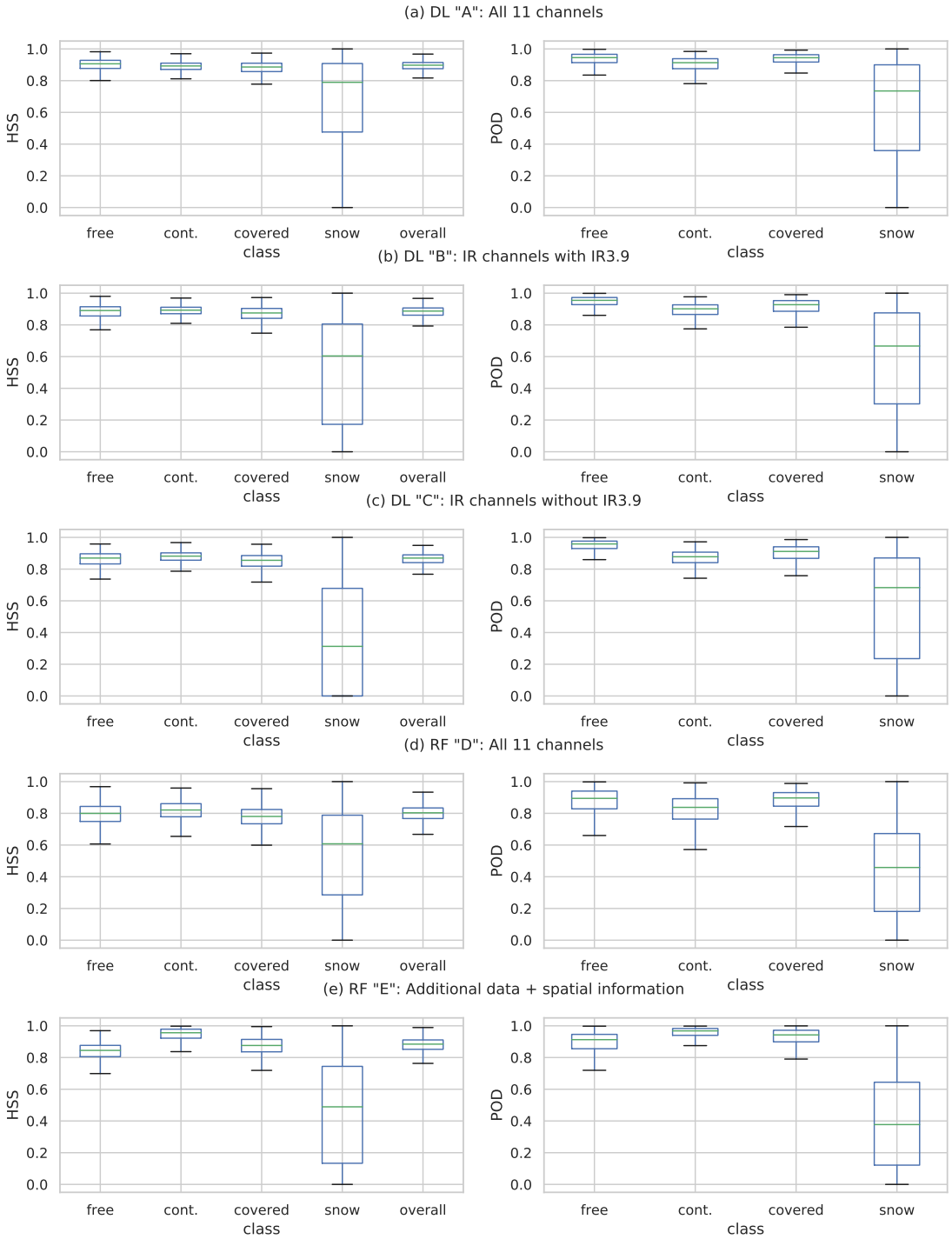


Figure 7.6: Box-plots of HSS and POD generated from all results of 7977 test scenes. HSS for all scene pixels is also included.

7.3.4 Seasonal and Diurnal Dependencies

The snow class in the CMa data depends on seasonality since there is less snow in Europe in the summer months. Additionally, the CMa algorithm uses the reflectance information from solar Channels 1–3 to classify pixels as snow-covered. Therefore, there are no snow pixels in the CMa data at night [DGF13]. The influence of this aspect requires further investigation. Figure 7.7 shows the HSS values grouped and aggregated by the month of the year on the left side and by the hour of the day on the right side.

For all CS-CNN scenarios, the seasonal variation of the cloud classes and the combined classes is quite low. All classes show values between 0.8 and 1.0, which are constant over the year, where Scenario A shows the highest values. However, the snow/ice-class reaches an HSS level of 0.9 during winter months, which is at the skill level of the other classes but drops to 0.4 during the summer months. A dependency on the hour of the day is also clearly shown by the right plot. In the night hours, no HSS is shown for snow/ice pixels while it rapidly rises (and falls) with solar irradiation between 5 a.m. and 6 p.m. to a level of 0.8.

Similar to the previous results, CS-CNN Scenarios B and C show slightly lower HSS values for all classes in comparison with CS-CNN A. While the HSS of the cloud classes and overall remains at a high level, significant changes are visible for the snow/ice class. Scenario B shows lower HSS and POD values, and Scenario C shows that the HSS level during summer months drops to 0.1.

The RF Scenario D shows seasonality patterns similar to the previous CS-CNN scenarios, but at lower value range. For all classes and overall, relatively constant HSS values between 0.7 and 0.9 are visible, except for the snow/ice class. For this class, we see a comparable skill during the winter months. However, it drops to 0.4 during the summer, similar to CS-CNN Scenario A. The diurnal performance distribution shows a similar pattern. At night, no skill for snow/ice is displayed, with solar irradiation, the skill rises to 0.7 and varies over the day. The patterns for the RF Scenario E show the most significant seasonal and diurnal dependencies, while still resembling the ones shown by the other scenarios. In contrast to the other models, this one shows clearly visible differences for all classes for daytime and month of the year. Cloud classes stay between an HSS of 0.8 and 1.0. However, they show more deviating value levels. The snow/ice-class again shows different patterns. While the performance in the winter months is similar to the other RF model, the performance drops to a value near 0.0 during summer. The diurnal distribution shows similar patterns, but snow/ice stays below an HSS of 0.6 during the daytime.

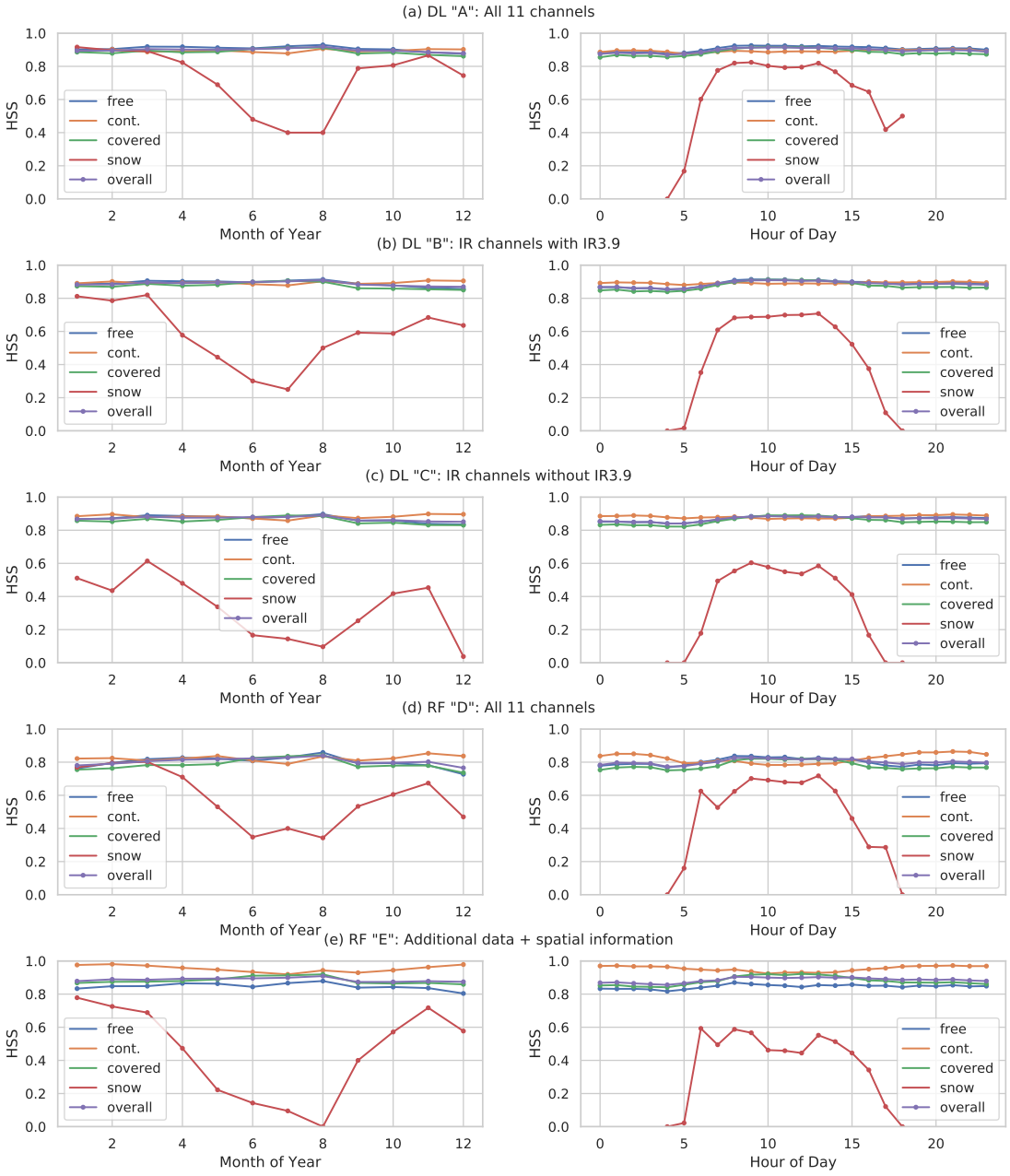


Figure 7.7: Plots of HSS median values from all results of 7977 test scenes grouped by month of the year and hour of the day.

7.3.5 Impact of Entity Boundaries

As shown in Figure 7.5, the CS-CNN segmentation shows deviations from the original cloud mask almost exclusively at cloud or snow entity boundaries (or edges). The RF classification in the same figure also shows a similar pattern, but there are also larger areas that do not match the CLAAS-2 cloud mask. To determine the actual effect of this error type, the evaluation of the results was modified in such a way that deviating pixels at entity boundaries are selectively excluded. Pixels at a boundary have at least one adjacent pixel with a different class. Now, we adapt the evaluation in such a way that we do not count pixels as wrong if there is at least a certain number of neighbors in the original cloud mask with the same class. If all eight neighbors of a pixel have the same class in the CMa, but the pixel class deviates from the CMa, it could be considered as noise. In the other cases, we can assume that the pixel is at the edge of an entity.

For the range of 8 to 1 required correct adjacent pixels, the evaluation was executed again for CS-CNN Scenario A. Figure 7.8 shows the resulting HSS values for the range of required correct neighbors. Reducing noise already shows an influence on the performance metrics of the segmentation. Excluding the pixels at entity boundaries from the evaluation clearly shows that the HSS increases above 98% for all classes. Even if only those pixels are excluded that have at least four neighbors with the same class in the original cloud mask, we already achieve a significantly better result.

7.4 Discussion

In this section, we discuss how CS-CNN copes with several issues compared to previous approaches for cloud classification.

The first issue of previous methods is their reduced usage of spatial information. This is different for CS-CNN, since it is based on U-NET, an architecture for image segmentation, as described in Section 7.1.2. Similar to U-Net, CS-CNN uses convolutional layers and learns to classify the multispectral SEVIRI data from spatial structures on multiple resolutions. Our results, presented in Section 7.3, show that CS-CNN produces excellent classifications with an overall HSS above 90% and accuracy levels above 94%. Looking back at the results of the specific models, clear patterns emerge. First, it is evident that using solar reflectance Channels 1–3 improves the results of the CS-CNN. While the cloud classes show excellent results in all scenarios and continue to improve with an increasing number

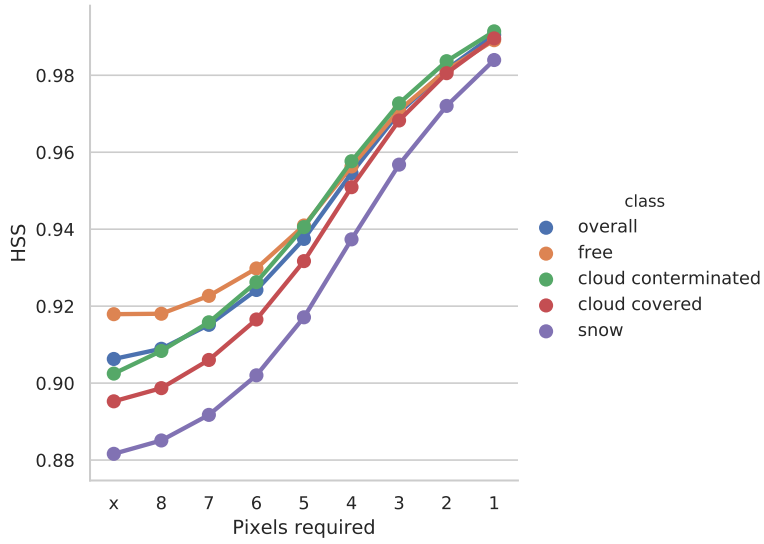


Figure 7.8: Impact of false classified pixels at entity borders. False pixels are ignored if the required number of surrounding pixels in the reference data have the same class.

of scenes for learning, the snow class is particularly striking. However, we should also note that the results for scenarios that do not include Channels 1–3, already show good POD values for the snow class. Confusion matrices generated over all scenes for CS-CNN (A) and RF (E) contain the distribution of undetected snow pixels, which is similar to the example scene. CS-CNN assigns them equally as cloud-free (45%) and cloud-filled (51%) while RF assigns the majority (67%) of them as cloud-filled. A look at the robustness of the models shows that the CS-CNN model using Channels 5–11 is already very robust. We improve the skill of all classes by adding solar channels, but only the snow class changes significantly. Therefore, we conclude that the IR channels alone are sufficient to generate robust cloud masks where the skill of each class is at the same level and has a small value range, as described in Section 7.3.3.

As discussed in Chapter 3.2, threshold-based and machine learning approaches require expert knowledge to identify relevant rules and features by hand. Productive cloud masks such as CLAAS-2 CMa use many rules to handle different characteristics of multispectral data where various parameters such as the position of the sun play a role in the classification. This is also the case for the FLS detection in Chapter 5 as well as for the random forest-based cloud detection presented in Chapter 6. The random forest used in our study learns a classification of cloud pixels from the multispectral SEVIRI data. However, as shown in Section 7.3, it

is not sufficient to use the satellite data alone. The first RF model shows lower HSS and POD values than the CS-CNN models. Only by adding additional data, handcrafted spatial information, and channel combinations, an RF model is generated that comes close to the results of CS-CNN. This is not necessary when using CS-CNN because CNNs learn the spatial features inherently. Moreover, in terms of robustness, we observed another strength of the CS-CNN models. The CS-CNN models provide very homogeneous classifications for all classes. This is different for the most advanced RF model. Although the feature selection for the model has led to improvements for the cloud classes, it considerably worsened the results of the snow class. This clearly shows that the use of the CS-CNN is considerably more robust than RF since the cloud and cloud-free classes show similar HSS values with small variability. Additionally, CS-CNN is not sensitive to common classification problems such as bright surface pixels (sand) and coastlines. Moreover, the application of traditional learning methods, such as RF, require considerably more expert knowledge in terms of handcrafted features and feature selection.

The third issue is the need to support nowcasting and processing of large time series concerning low latency and fast processing, respectively. Deep learning frameworks, such as Caffe, speed up CNN execution by implementing the different layers for execution on GPUs. Thus, the runtime required to classify a single scene is very short (within milliseconds), as shown in Section 7.3. The proposed CS-CNN classifies an input scene instantly (25 ms) on a GPU. Execution on a CPU requires a significantly longer time (1.24 s), and the classification using RF takes 1.86 s which includes the time to generate spatial information and combine channels. Additionally, RF can handle large amounts of data based on its sampling approach. However, this means that large amounts of training data is never used.

Despite its excellent performance, one of the important requirements of CS-CNN is the availability of ground truth information for the learning phase and the performance evaluation. In particular, a public dataset with only correct labels of clouds is not available yet. Additionally, labeling the amount of data used for this study manually is infeasible. We circumvented the problem by using a well-validated dataset, CLAAS-2 CMa, as reference data for CS-CNN. Note that CLAAS-2 CMa suffers equally from the challenges presented in Section 7.1 for the SEVIRI data. Our study also shows that a large amount of training data constantly improves the accuracy of the model. However, it can be seen in Section 7.2.1 from the training progress that even small amounts can produce accurate models. This can be complemented by technologies for data augmentation as used by U-Net [RPB15]. Data augmentation generates additional training data, e.g.,

through distortions or rotations of the available data.

An interesting aspect is whether the use of a different classification technique can identify relationships that are not explicitly specified. We used CLAAS-2 CMa for training, which uses threshold-based methods. These require the VIS channels to detect snow in the SEVIRI data. We know that the performance of CS-CNN for the snow class depends on the time of day, as shown in Section 7.2.2. As expected, CS-CNN can detect the snow class better if the solar channels are included. However, this only means that the CNN can resemble the CLAAS-2 CMa better and not that the classification of the real snow cover is actually more correct. There is only snow if the solar channels are not dark. Obviously, CS-CNN can learn correlations that are only implicit in the original data. This is evident from the fact that the snow areas can be well classified from IR data alone. The median FAR values for the snow class of the IR only CS-CNN models are 45% and 36%. However, the CLAAS-2 CMa does not contain snow in night scenes, so even a correct detection of snow would be considered wrong and increase the FAR in these cases. We have observed scenes in which CS-CNN has detected obvious snow areas even if the original does not include them. It is, therefore, possible that the CS-CNN classification can perform better than the original in some aspects . For a more detailed investigation, however, real ground truth data are required.

Future work could enhance e-research systems/labs such as the VAT system [Bei+17c] to combine CNN training and execution with workflow-based data handling, preprocessing, and visualization of results.

7.5 Conclusions

In this chapter, we present CS-CNN, a novel cloud mask generation approach using a CNN with a segmentation architecture. We show that CS-CNN is very well-suited to process multispectral remote sensing data and to classify clouds. Compared with a frequently used RF approach, CS-CNN requires significantly less expert knowledge and effort. When compared with RF, the results of CS-CNN show higher accuracy, are produced faster, and are significantly more robust, i.e., they are more consistent. CS-CNN provides high overall accuracy (0.94) and HSS (0.90) values and requires only 25 ms of computation time for classifying an input scene using a GPU.

8

The Visualization, Analysis and Transformation System (VAT)

The previous chapters present the processing of a large raster time series and methods to detect clouds and FLS in MSG data in particular. Remote sensing data can benefit biodiversity research, e.g., to investigate the correlation of migration and weather [Tur+03]. However, the challenges of handling and integrating remote sensing data are immense for non-experts. Thus, researchers in geosciences require new systems to handle heterogeneous, large, and complex data. This is one requirement for innovative ideas and problem solutions [Gil+18]. Therefore, this chapter presents the Visualization, Analysis, and Transformation (VAT) [Bei+17b], developed at the Database Research Group at the University of Marburg. VAT enables web-based interactive data visualization and exploration to facilitate access and analysis of spatio-temporal data. It provides operators for processing and combination of raster and vector data. Additional special-purpose operators for MSG calibration allow non-experts to handle the MSG time series.

First, this chapter motivates the necessity of the VAT System. Then, the components of VAT, the back-end MAPPING (Marburg's Analysis, Processing and Provenance of Information for Networked Geographics), and the web-based front-end called WAVE (Workflow, Analysis and Visualization Editor) are presented. We use real-world use-cases to present compact overviews of the design and techniques used in both components. Finally, a use-case presents the combination and analysis of spatio-temporal plane trajectories and the cloud mask generated from MSG using the CS-CNN as well as weather radar data.

8.1 Motivation and Background

Spatio-temporal big data is collected by multiple sources and the amount, as well as an exponential growth of data [MG15], makes a data-driven research approach increasingly popular [AAG03; Ste+13]. Data-driven research focuses on the exploration and visualization of existing data to find interesting patterns, correlations, and anomalies. However, exploration and analysis to identify exciting correlations and anomalies require software designed for this task.

Biodiversity has an additional problem, namely the amount of unused data [CW14; Gri15]. Specialized repositories exist for collection of different kinds of biological data. Movebank [Kra+11] for example, is a repository for animal tracks. However, repositories to collect all biological data are required to avoid data loss and assure long-term data availability.

The **German Federation for Biological data (GFBio)** [DGG+14] is an infrastructure project to provide a central instance where research data can be stored, and existing data is accessible. Multiple partners, including research collections, data centers, and archives, provide expertise and services in GFBio. The GFBio portal[1] provides a central instance for data submission and data access. Data curation and support for data management planning facilitates data submission and re-usability. A search index is built by harvesting the data in the GFBio data centers and collections. The GFBio portal's search uses this search index to provide a faceted full-text search with semantic and ontological enhancements to find data [Löf+17]. Biodiversity data is usually geo-located and provides temporal information, e.g., when it was created or when a reported observation was made. To visualize the spatio-temporal data on a map, a connection to VAT was developed. This connection enables users of the GFBio portal to search for data and visualize the results on a map. Additionally, VAT allows to explore and analyze correlations with other spatio-temporal data.

VAT was developed to enable different kinds of users to explore, combine, and correlate spatio-temporal data in an exploratory fashion [Bei+17b]. It is a web-based system which reduces the initial complexity of access to spatio-temporal data and relives users from handling of large and heterogeneous spatio-temporal big data. This is facilitated by enabling connections to data sources, e.g., GFBio to access biodiversity data, and providing relevant data, e.g., from the Global Biodiversity Infrastructure Facility[2] (GBIF), which provides observations of animals and plants. VAT uses GDAL to access raster data and provides support

[1] www.gfbio.org
[2] www.gbif.org

for time series. It also provides various sources for vector data. The web-based map presents all data as layers and while the data from multiple sources might have various CRSs, VAT handles re-projections and enables combined visualizations.

VAT aims to benefit multiple types of users. Researchers, e.g. from biodiversity, are often interested in environmental data to model distributions, movements, or habitats. However, handling large raster time series and converting CRSs often requires expert knowledge and preprocessing to generate usable data for existing software. Computer scientists program efficient and fast processing systems but often lack knowledge about spatio-temporal data and knowledge from the remote sensing and biodiversity domains. Geographers, as the domain experts, are challenged by big data problems. The data is large (volume), e.g., satellite raster time series, heterogeneous (various), e.g., raster and vector data is different projections, and the growth of data and data sources keeps increasing (velocity).

To enable explorative research, and interactive analysis, which is crucial for geo analysis tools [Rot13], VAT uses different techniques for efficient raster access and accelerates processing with OpenCL. The user interface WAVE provides visualization and operators for exploration and analysis, following the visual information seeking mantra by Shneiderman [Shn03].

In the following, we introduce the architecture and concepts used by VAT. Then, aspects of back-end and front-end are explained using example applications. Last, we present two use-cases. In the first one, we adapt VAT to present a single spatio-temporal application. Second, we connect VAT and and a system for event processing to combine raster data and trajectories of planes.

8.2 VAT - Design and Architecture

VAT consists of two parts, a back-end for data processing and a front-end to support interactive visualization and analysis with a concept called exploratory workflows. Figure 8.1 shows a schematic of the components as well as connections to data sources and users. On the right, the data sources are depicted. VAT handles raster and vector data from local as well as remote sources, e.g., the GFBio portal. The back-end, called MAPPING, handles the processing through operators which are composed of a requested processing description, represented as a workflow-graph. Statistical operations are mainly implemented with R code. An additional R-server component can handle the raster and vector data structures

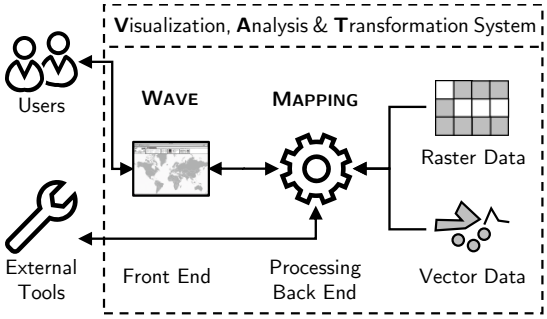


Figure 8.1: The VAT Systems components WAVE and MAPPING. MAPPING connects to the data sources and provides processing. WAVE uses OGC protocols to request data from MAPPING. WAVE presents data on a map as well as via table and plots [Bei+17b].

used by MAPPING. The R-server returns raster and vector data in compatible data formats as well as plots as images and summaries as text. An R operator wraps this functionality and allows to use it in the processing description. This way, it is possible to express functions, which are not yet implemented as special operator, with R code. While MAPPING provides access via WMS, WFS, and WCS protocols, most users want to use the visual user interface WAVE. WAVE presents the spatio-temporal data requested from MAPPING on a map, vector attributes as a table, and plots as images. More details are explained in the following using examples of explorative workflows.

8.2.1 Explorative Workflows in VAT

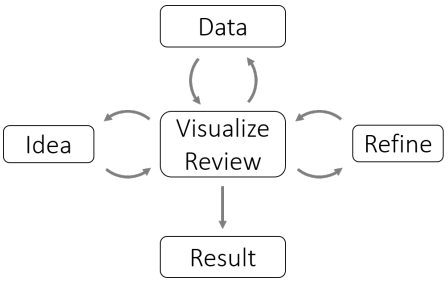


Figure 8.2: The explorative research cycle. A scientists starts by exploring data or an idea. By adding more data, analysis and refinement, as well as changing the visualization new ideas are formed and finally a result is created.

Explorative workflows are the concept used in VAT to support the data-driven research cycle. This concept follows the visual information seeking mantra by Shneiderman [Shn03]. Figure 8.2 shows this graphically. A user starts with a selected set of data. If there is an interesting pattern or phenomenon, visualization and analysis tools provide ways to explore and explain this in detail. Eventually, additional data is added and correlations are identified. Iterating this process, of visualization, idea refinement and analysis, a new scientific result is finally created.

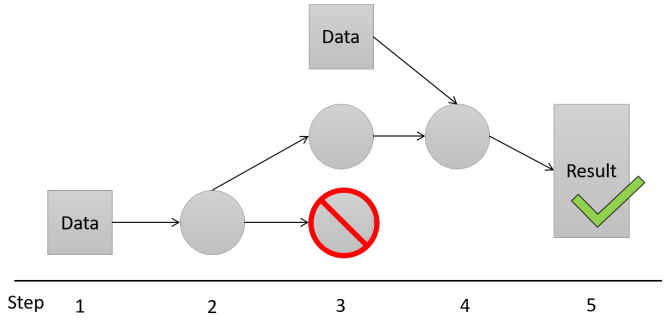


Figure 8.3: An explorative workflow. After three steps, the path at the bottom produced an unsatisfying result at step 3. Therefore, new data was added and different path leads to a result after five steps.

The process of finding and creating new results is presentable as a workflow. This is basically a flattened view of all steps and inputs used to generate a result. As some ideas, data integration steps, or analysis results might be unsatisfying, the workflow view shows dead-ends. Figure 8.3 presents an example of explorative workflows, which shows the processing steps from left to right. The explorative workflow approach presents obstacles in most GIS, e.g., in a desktop GIS, where users have to remember and write down all steps to be able to recreate a result. VAT provides a solution to this problem. All steps composed and executed in the interactive user-interface WAVE are recorded and are stored as a workflow automatically. This way, analysis and exploration are reusable and understandable.

8.2.2 Realizing the MSG Data Preprocessing

MAPPING is implemented with modern C++. It uses GDAL for data access as well as other libraries, e.g. GEOS[3] for vector operations. Mapping supports both

[3] `trac.osgeo.org/geos`

spatio-temporal data models, i.e., raster and vector data. Additionally, statistical output like histograms, scatter plots, and text are supported. VAT aims to support data-driven research with explorative workflows. Therefore, composing the available operators is a key feature.

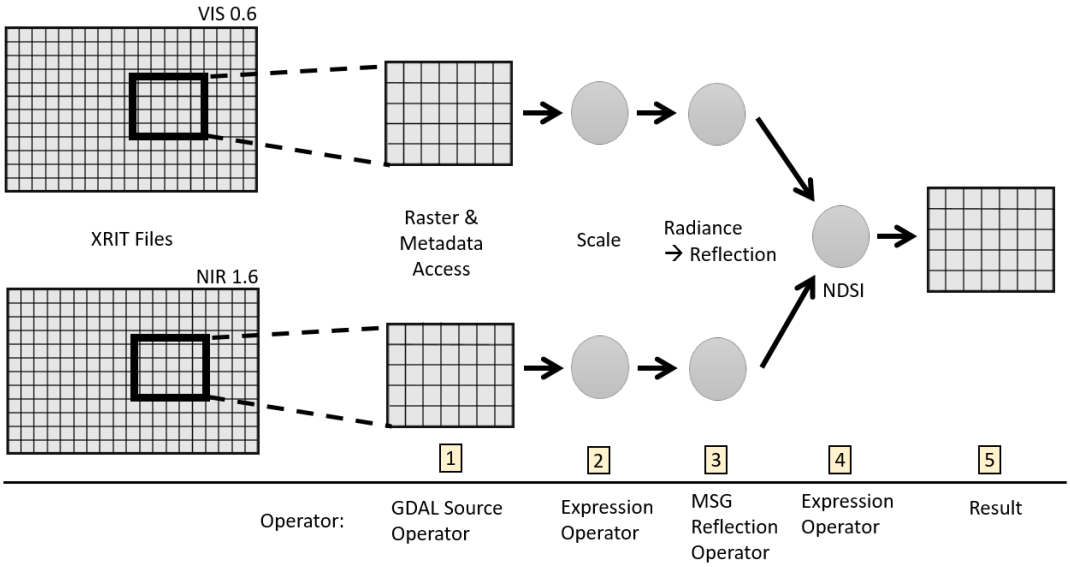


Figure 8.4: The workflow for processing MSG raw data into reflectance values and calculation of the NDSI for snow detection.

To describe the components of MAPPING and the processing approach, describe the MSG data preprocessing, introduced in Chapter 4, in VAT. Additionally, the NDSI introduced in Chapter 2.4.2, is calculated from two MSG channels.. As discussed, MSG raster data is a long time series, with a new scene for each 15 min. Each scene contains 11 channels in the infrared and visible spectrum and each raster is 3712×3712 pixels in size. We assume that a user wants to calculate the NDSI for Europe (767×510 pixels) at an arbitrary but fixed time interval. For the sake of simplicity, we assume that this time interval is 15 minutes long and covers the temporal validity of exactly one MSG scene. For this purpose, a workflow is assembled, which is indicated in Figure 8.4. A query to MAPPING always consists of a workflow coded in JSON and a query rectangle that defines a spatial bounding box and a time interval. This is further referred to as *spatio-temporal query rectangle*. MAPPING de-serializes the workflow from the JSON data, i.e. it generates all operators and concatenates them as an operator graph. The calculation of a result begins with the transmission of the requested spatio-temporal query rectangle through the graph to the data sources. Individual operators can also change the requested bounds. This is necessary, if a filter, e.g., for edge de-

tection (see Chapter 2.5.2), needs to access neighboring regions. Therefore, an operator increases the query rectangle by the required number of pixels in each dimension.

Figure 8.4 presents the MSG raster processing workflow. The individual steps are numbered from (1) to (5). First, a data source (1) loads the raw MSG data and creates a raster object. One data source in MAPPING is the GDAL operator. In this case, it loads the raster data stored in the HRIT format from the hard disk. GDAL takes care of data formats and subsets the large raster to the correct size. This uses the requested spatial bounds which are mapped to pixel coordinates. The XRIT data for each scene is split into 8 tiles per channel, GDAL loads only the necessary tiles, and then subsets and combines the pixels from the tiles. The XRIT file names contain the creation time of each MSG scene. Since the time series has uniform steps, a data description containing the file name pattern allows MAPPING to open the correct files. GDAL also enables efficient access to raster data through tiles and pyramids (see Chapter 2). Since a query in MAPPING also contains the pixel size of the requested raster. If a pyramid is available for the raster data, GDAL loads the pyramid level with the matching resolution. The GDAL operator in MAPPING maps the requested time to a corresponding file. To convert the MSG raw data into reflectance values (2), the *offset* and *slope* metadata is required. MAPPING employs GDAL to load this information from the data sources and transfers it to the raster object, which contains the pixel values as well as metadata. The result of the operator is passed to a special operator for MSG data calibration.

MAPPING contains three ready to use operators for the MSG calibration steps presented in Chapter 4. Operators provide functionality to transform raw data into radiance, reflectance, and brightness temperatures. The implementations use the OpenCL code developed in this thesis. Each OpenCL kernel is executed on multi-core or many-core systems in parallel for all pixels.

Calculation of the NDSI requires MSG data from Channels 1 and 3. The reflectance operator uses the direct conversion approach presented in Chapter 4 to transform the raw data provided by the GDAL operator. First, a mapping from 10-bit values to reflectance is generated. Then the raw data is directly transformed into reflectance values (3). The creation time of each scene plays an important role here. For calibration, the appropriate SZA for each pixel is required. The operator uses the time from the spatio-temporal query rectangle as the time the MSG scene was created. It calculates the coordinates as well the SZA for each pixel in parallel to transform the radiance to reflectance values.

The next step is to combine the calibrated reflectance values and calculate the

NDSI (4). Since the calculation rule for NDSI is straightforward (see Chapter 2.4.2), this step also uses the Raster Expression Operator with two inputs. Operators in MAPPING can have an arbitrary number of inputs, but only one output. If data with different spatial or temporal bounds are combined, operators for homogenization must be inserted into the workflow to adapt the resolution or the projection. WAVE is able to insert the homogenization operators automatically, therefore users are not required to do this manually. Using OpenCL, the expression operator calculates the NDSI in parallel for each pixel. This represents the result (5) of the workflow and the answer to the query. MAPPING stores workflow results either as a raster file on disk or returns it as response to a WMS, WFS, or WCS request.

While this example only presents raster data processing, the functionality is similar for vectors. With vectors, however, operators consume and produce sets of features, where each vector can have its own temporal validity. This is not the case with raster data. Operators consume and produce raster objects, which are valid for a specific time interval. This corresponds to the snapshot approach used by many GIS.

8.2.3 WAVE

In this section, the concepts of WAVE, VAT's web-based interactive user interface, are explained using a biodiversity workflow. We assume a researcher collected information about forest elephants and wants to compare them to African elephants. Figure 8.5 presents a screenshot of WAVE. The central element is the map view, in which several data layers are visualized. On the left side of the map, the layer list shows all currently available raster and vector layers (A-E). These are displayed from bottom to top on the map, whereas upper levels can cover lower ones. On the map the layers from the list as well as a background, which usually displays OpenStreetMap[4] data for orientation, is presented. The layer at the bottom (B) shows the forest cover in percent. Hansen *et al.* [Han+03] have calculated the global forest cover for the year 2000 from MODIS data. They also provide the forest change for the years 2000 to 2012 as a raster time series. In the layer list, a suitable legend for this type of data is presented. The second layer is a vector layer, which presents observations of African elephants (B), which were exported from GBIF. Layer C contains the observations of forest elephants the researcher gathered, e.g., as a CSV file. VAT enables users to import personal data, in this case the data was also gathered from GBIF. The study area the researcher is interested in is represented as Layer D. On the map, a yellow-colored outline visualizes

[4] `openstreetmap.org`

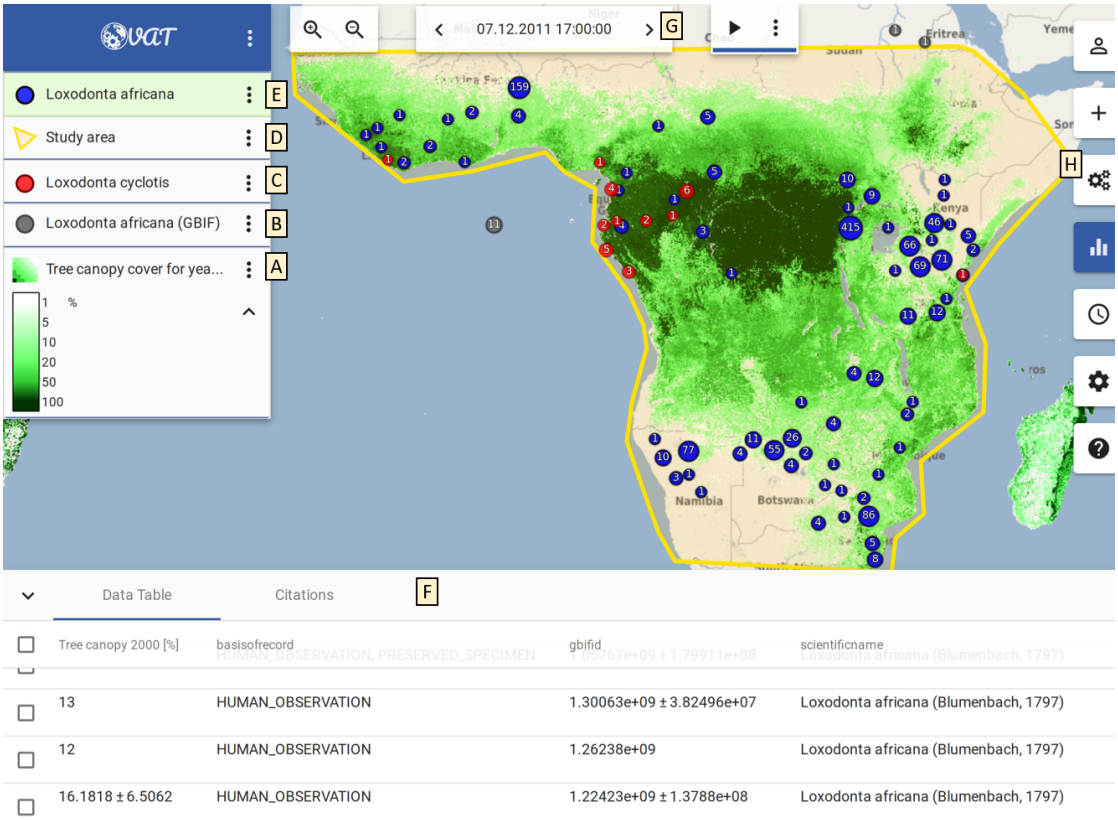


Figure 8.5: VAT's interactive user interface WAVE. The central component is the map view. The layer list shows all layers (A-E) visible on the map. Attribute values and data lineage is presented at the bottom (F). At the top, the time selector (G) allows to change date and time of the displayed data. A menu on the right (H) enables users to add new layers and to apply operations.

the study area. WAVE provides a mode to create new vector layers and allows drawing new features interactively. The combination of Layer B and D is presented as Layer E. WAVE provides functionality for interactive data combination. The resulting layer only contains the observations within the study area. Note that the gray point cluster in the ocean is not part of Layer E.

The lower part of the figure depicts the data table (F), which currently shows the attributes of the African elephants (E), as plotted on the map. At the top mid of the screen, a selection (G) for the visualization of time or time intervals is shown. Finally, on the right side of the figure, WAVE's menu bar (H) is located. It contains different options to add new data, select operations or display statistics.

All displayed map layers, their attribute tables, as well as the plots are described by workflows. User actions in WAVE are chained together to a workflow that is serialized as JSON. WAVE sends the workflow as a JSON object to MAPPING and receives data to display. In general, WAVE remembers for each layer how it was created, i.e., which operations were executed to create it. This allows WAVE to extend all workflows and to request them for changing temporal and spatial bounds. This workflow approach enables interactive exploration and analysis, which is very important for a web-based GIS for spatio-temporal data.

WAVE is built on top of the web-map library OpenLayers[5]. Since the map can be panned interactively, it does not make sense to recalculate all raster data at once and for each movement. Openlayers therefore splits the visible map area and queries tiles with a size of 256×256 pixels. For a 1920×1080 pixel monitor, there are 32 tiles. Openlayers requests tiles via WMS from MAPPING by sending the workflow in a JSON format. It also passes the time interval which is given by the time selection (G) to MAPPING. MAPPING instantiates all operators from the workflow. The fixed size of the tiles and the transferred spatial and temporal dimensions are passed through the operator graph. A source, e.g., GDAL, opens the relevant files or connects to a database. Using the tile size in pixels, GDAL can determine from which pyramid level data must be loaded. If the map is moved or the selected time is changed, the same workflow is used again to load missing tiles.

Figure 8.6 shows the workflow representation of all layers created for the elephant example. First, the elephant expert adds the forest cover raster dataset [Han+03]. This is step (1) in the workflow and Layer A in Figure 8.5. Then the African elephants are added from the GBIF data provided by VAT (2,B). Now, the self-observed forest elephants are added from a CSV file (3). For this purpose, WAVE transfers the content of CSV files to MAPPING, which stores it in a PostGIS database. The *Raster Value Extraction* operator (4) combines the forest cover pixels and coordinates of the elephants. The result is an enhanced layer (C), where each observation of forest elephants has the raster value as a new attribute.

Since Layer C contains obviously wrong data, e.g., elephants in the ocean, the expert adds the study area. WAVE enables users to draw new shapes, which are then stored by the back-end. A matching source operator (5) provides access to the custom data, which is displayed on the map as Layer D. To remove the locations outside the study area, the *Point in Polygon* operator (7) performs a join between polygons and points and returns only the points within the polygons. Again, the

[5] `openlayers.org`

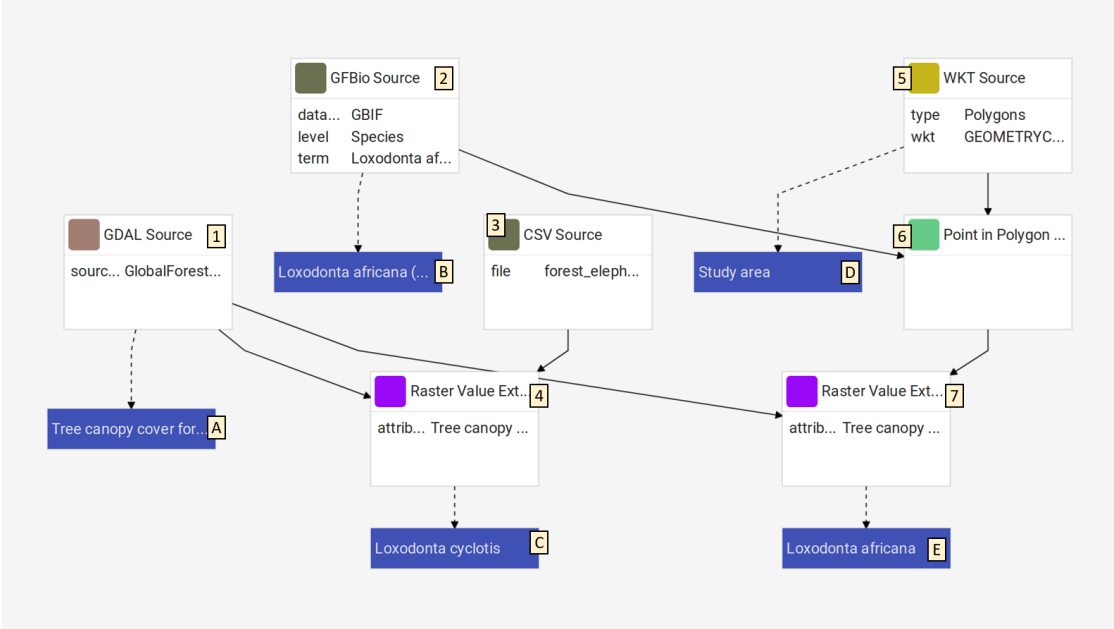


Figure 8.6: The workflow used to generate the layers in Figure 8.5

forest cover values are added to the points (7) and the result, Layer E, is added to the map.

The difference in the distribution of both elephant species is also visible in Figure 8.7, which shows histograms generated from the forest cover values at the coordinates of elephant observations in Layer C and Layer E.

8.2.4 Use-Case: Clouds, Rainfall and Plane Trajectories

Complex Event Processing (CEP) [Krä07] is a technology for continuous monitoring and analysis of event streams. One of the main functions of CEP is to support pattern matching queries to detect user-defined sequences of predicates on event streams. Optimizing queries to deliver the desired results, however, is challenging. Therefore, we have built a data source in VAT, which allows to query events from the CEP using JEPC [Hoß+13] and ChronicleDB [SS17]. In the presented use-case, the events are aircraft trajectories from the OpenSky-Network [Sch+14]. This allows users to interactively study aircraft motion and integrate raster data, resulting in insights beyond event-based analysis.

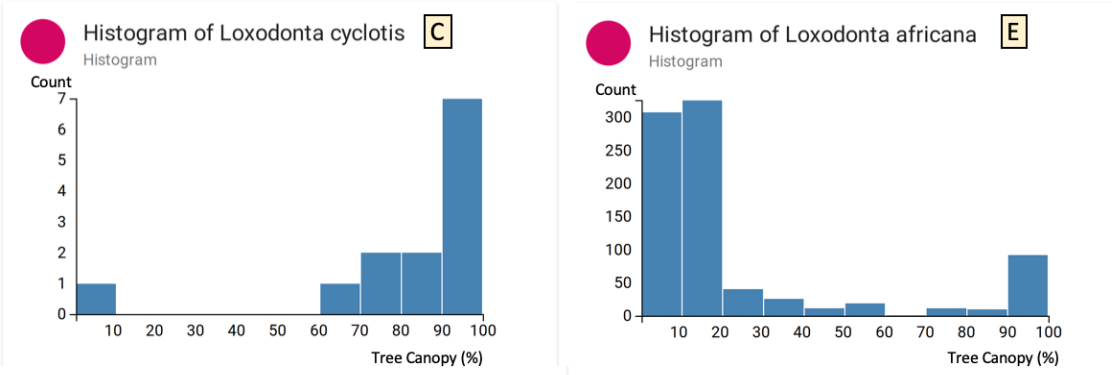


Figure 8.7: Histograms for African (E) and forest (C) elephants. The bars show the count elephant observations grouped by forest cover values with bin size of 10 percent.

CEP systems offer many functions on data streams, one of the most important is pattern matching. The primary dimension of CEP systems is time. Patterns are formed by a sequence of predicates. However, events with spatial extension are of particular interest. Since events contain attributes, they can also represent spatial information such as location, altitude, and speed. For example, a pattern for recognizing landings can be described for airplanes but also for birds. If object decreases its speed and reduces its height above the ground in a sequence of events, then one can assume a landing approach.

JEPC is a middleware that offers a uniform processing of events. Two components of JEPC are relevant for the presented use case: First, the event store ChronicleDB and second special operators for spatio-temporal data that support pattern matching [Kör+19].

ChronicleDB serves as a buffer for event data in this use-case. Through indexes and the support of pattern matching, incoming data streams can be stored and replayed on request. This is important because VAT allows users to change the time to explore temporal changes. The requests formulated as part of an explorative workflow must therefore be able to be repeated. MAPPING passes the spatio-temporal query rectangle and a request to ChronicleDB via a REST interface. The output is spatio-temporal data serialized as JSON. Then the GDAL operator is used to read the serialized data. In WAVE, a new operator was built which supports expressing queries for JEPC in its SQL-like query language.

Data

This use-case combines plane trajectories from the OpenSky network with the cloud masks generated in Chapter 7 and additionally with weather radar data.

The OpenSky network [Sch+14] is an open infrastructure that collects freely receivable flight data. In order to improve aviation safety and management, aircrafts must transmit their position using Automatic Dependent Surveillance Broadcast (ADS-B). This protocol is not encrypted and can be received via a Software Defined Radio (SDR) on a frequency of 1090 MHz. Low-cost hardware, e.g. a Raspberry Pi, and open software, such as *dump1090*[6], but also commercial of-the-shelf solutions, allow anyone to observe aircraft in their vincinity. The OpenSky network operates servers on which many volunteers transmit the received data. Thus, it enables a large-scale real-time monitoring of flight movements and the creation of corresponding databases. The OpenSky network also makes it possible to request and use historical positions and trajectories. We have downloaded the flight events consisting of position, time and other attributes such as speed, altitude and aircraft ID for February 4, 2019 and inserted them into ChronicleDB. In total, this dataset consists of about 34 million events. At the time of writing, there were over 16,744,000,000,000 ADS-B messages stored in the OpenSky network. The message rate observed by the network is approximately 300,000 events per second.

In aviation, as in all areas of traffic, weather information is very important. In Germany, the German Weather Service operates a radar network that enables the precipitation rate to be identified with a very high resolution from the reflected signals [Bar+04]. While the DWD's Open-Data Server[7] provides hourly data for larger periods of time, 5-minute intervals are only available for two days after the measurement was created. Different production stages of radar data are made available as raster time series. For our use-case we collected a radar product, called *WX*[8]. It represents the reflectance of the radar signal for each pixel of a grid covering Germany [Wet16]. This product has a temporal resolution of 5 minutes and a spatial resolution of 1 km×1 km. The resulting rasters are therefore 1100×900 pixels in size. We use the *wradlib* [HJP13] Python library to read and transform the binary reflectance rasters.

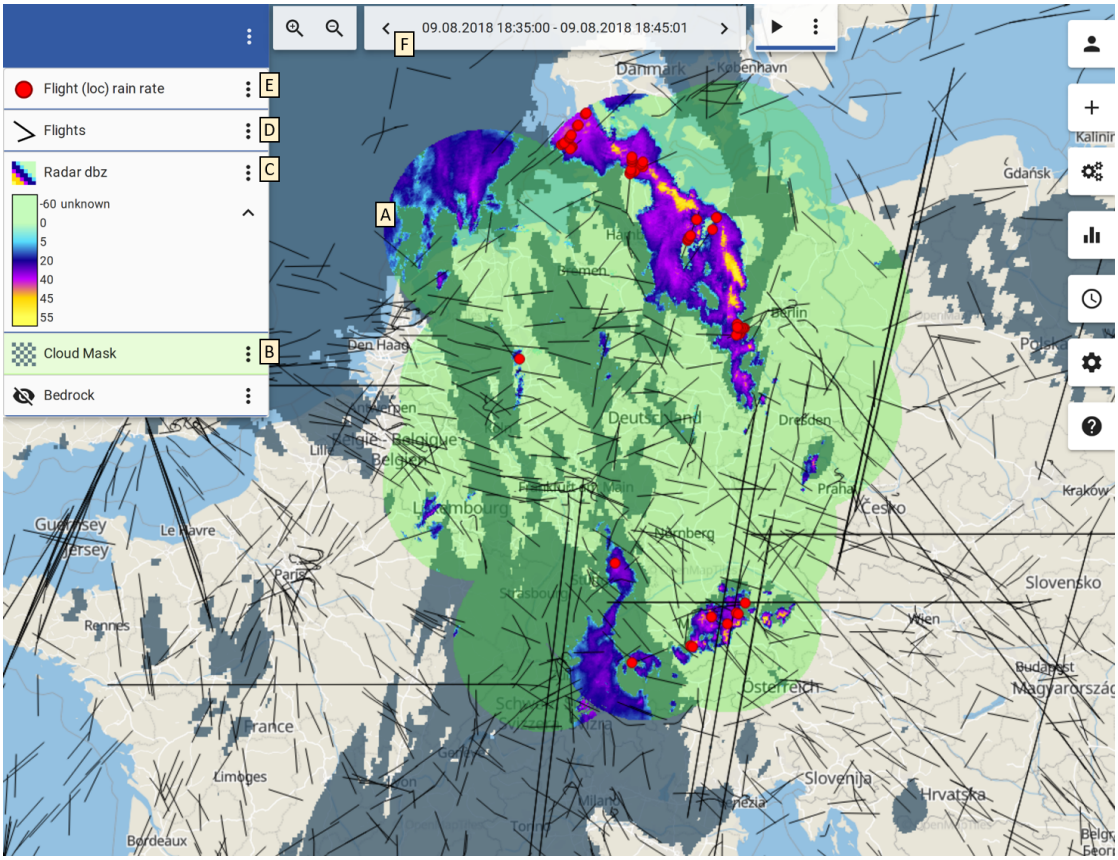


Figure 8.8: A combination of plane trajectories, cloud mask, and weather radar. Plane position in areas of severe weather are indicated by red points.

Spatio Temporal Event Enrichment

Figure 8.8 shows WAVE, the user interface of VAT, as described in Section 8.2.3. The first noticeable change is the type of the selected time (F). In this case, a time interval of 10 min starting at 2011-08-09 18:35 UTC is chosen. The map (A) displays four layers, which are also represented by the layer list (B-E). The bottom layer is the cloud mask (C). The CNN framework developed in Chapter 7, calculates the cloud mask and stores it as raster files. The GDAL operator is able to access and load the generated cloud masks from the raster time series. The second layer (D) is the DWD radar data. While the cloud mask is styled as gray cloud areas, the radar reflectance measured by the radar in dBz is col-

[6] github.com/antirez/dump1090
[7] https://opendata.dwd.de
[8] opendata.dwd.de/weather/radar/composit/wx/

orized. The mapping from reflectance to rainfall is a linear function. Therefore, the reflectance visualization correlates with rain rates [Bot16]. Light blue indicates low rain rates and a gradient from purple to yellow indicates higher rain rates.

The two layer at the top (D) and (E) presents flight trajectories and positions from the OpenSky data. The time selector (F) reveals, a selected time interval of 10 min. This information is part of the request send to MAPPING and it is passed on to ChronicleDB, that stores the flight positions of the planes.

For Layer E, ChronicleDB returns all positions of planes, which are inside the requested spatial and temporal bounds. For Layer D, it groups all positions by time and plane identification into trajectories, which are symbolized as black lines on the map.

Looking at the map (A), clusters of air traffic are visible in London, Paris, Amsterdam, and Frankfurt. Cloud formations are visible in central Europe. A big cloud band spans from the south to the north. In the radar data layer, a green color indicates radar coverage and the blue-purple-yellow gradient indicates the rain rate. In particular, areas with a high rain rate, which correlate to severe weather situations, are visible over Northern Germany. In order to identify flight positions within cloud-covered areas with high rain rates, the user incorporates VAT's visual analytics toolbox. For this task, the individual flight positions, which form the trajectories, are added as a point layer to the map (F). The raster value extraction operator combines the weather information from the raster images (cloud cover and rain rate) for each flight event (i.e. position). To identify the locations and inspect patterns, users can colorize the point layer based on attribute values. To generate a new layer with only the interesting events, the filter operator takes a user defined threshold and creates a layer with only the relevant data. Layer F requires the cloud class to be either cloud-contaminated or cloud filled. Additionally, the radar reflectance is set to be over 40 dBz, which corresponds to heavy rain.

This use-case shows additional interesting patterns in the trajectories. A lot of bad weather locations are located near Munich. The visualization shows that one cannot always avoid bad weather situations to reach some airports. To avoid the areas with bad weather, some trajectories show avoidance strategies, e.g., direction changes around rain areas.

8.3 Conclusion

In this chapter, the Visualization, Transformation, and Analysis (VAT) System was presented using various use-cases. VAT provides a unique combination of functions not only for biodiversity and weather information, but mainly for linking heterogeneous spatio-temporal data. Efficient access to raster time series, interactive visualization with explorative workflows, as well as the linking of raster and vector data, offers various user groups the opportunity to gain new insights.

By combining our visual analytics platform with a database-driven event processing system, our approach demonstrates distinct advantages. VAT incorporates time as an integral dimension. This makes it possible to provide access to spatio-temporal data for explorative research. In addition, it is possible to recognize and examine patterns that would not be recognized as easy in other systems, which often ignore the temporal dimension or focus on either raster or vector data.

For future work we want to extend the features of VAT by creating a connection to frameworks for deep-learning. One example is the classification of clouds, which should will be done via a new operator in VAT. The CNN approach, presented in Chapter 7, allows instant classification into classes represented as raster or polygons. This facilitates exploration and combination with other data types. We also want to increase the interaction between VAT and event processing. In combination with instant cloud classification, this will allow to monitor patterns in trajectories and their correlation with weather data.

9

Summary, Conclusion and Outlook

This chapter revisits and evaluates the hypothesis from Chapter 1. It concludes the thesis with a summary and an outlook on future work.

9.1 Summary and Conclusion

Spatio-temporal raster data produced from remote sensing sensors on satellites is important for many domains. Clouds have a crucial role since they are both study object and obstruction when focusing on the earth surface. Geostationary satellites, e.g., MSG, are essential for weather prediction and climate monitoring. The lack of long term time series for FLS, as well as the requirement to reduce required expert knowledge for the creation of new satellite data products, was the primary motivation for this thesis. Additionally, domains like biodiversity and traffic rely on fast or long-term cloud data. To gain information from raster time series, e.g., cloud masks, a visualization, and analysis system with time as an integral component was required. The central contributions of this thesis are:

1. This thesis presents a new processing chain for computing FLS masks for long time series of MSG data. The presented approach merges and adapts existing schemes for FLS detection at day and night and uses efficient, parallel, and concurrent computing to process a whole MSG scene within seconds on commodity hardware.

2. The second contribution is a novel cloud classification method using a CNN architecture for image segmentation. All pixels are classified simultaneously instead of individually within a fraction of a second. Feature engineering is not required for the presented approach, which outperforms state-of-the-art methods like RF.

3. It presents the concept and examples for the integration of raster time series into VAT, a web-based system for exploratory processing and linking of spatio-temporal data. It enables the combination and correlation of heterogeneous spatio-temporal data and facilitate the creation of new insights.

To enable handling and analysis of large raster time series, all tasks conducted in this thesis use concepts for efficient and parallel processing. This is related to Hypothesis **H1**, which was formulated as follows:

H1 Efficient, parallel, and concurrent methods enable processing of large MSG time series with very low processing times.

The processing of the large MSG SEVIRI raster time series for the generation of a ten-year FLS climatography, presented in Chapter 5, uses efficient data loading, parallel processing on GPUs and CPUs, as well as concurrent execution of subtasks. The processing time of a single full disk MSG scene is only 2.5 s. This is significantly lower than the 15 min temporal resolution of MSG. Using GPUs for the cloud segmentation, the CS-CNN enables the classification of a MSG scene for the selected study area in 25 ms. Additionally, the realization of the MSG chain in VAT allows non-experts to use extensive raster time series data in an interactive fashion and combination with other datasets. Therefore, **H1** was confirmed.

The processing of MSG data to create a large FLS time series required unification of existing approaches for day and night [CB07; CB08]. The opportunity of a re-implementation for a homogeneous classification and dynamic study areas was also sized to add efficient, parallel, and concurrent processing. Hypothesis **H2** was formulated as follows:

H2 A unified and area independent spatial partitioning strategy enables homogeneous FLS classification for large time series and parallel processing.

The method created by merging and adapting existing schemes for FLS detection at day and night allows the production of one homogeneous FLS dataset. The FLS detection steps make use of tiles and overlapping windows for spatial partitioning. This enables results independent from the total study area and allows processing of other and larger areas as well. An efficient data loading module based on GDAL's MSG driver is one part of the modular designed processing tool. Partitioning the data into tiles allows a parallel processing scheme. Additional, implementing raster processing with OpenCL permits intuitive single-pixel operations as well as complex operations like the parallel histogram generation. The best performance was achieved when using both CPU and GPU together. The processing chain runs different processing steps concurrently on CPU and GPU, which provides

additional speedup. The generation of a ten-year FLS climatology for Europe confirms **H2**.

The MSG data processing chain also produced the training and evaluation data for the CNN-based cloud segmentation method. Current cloud classification methods use either sets of rules and thresholds or shallow learning methods to classify individual pixels. Additionally, domain experts have to engineer and select relevant features. Therefore, a significant degree of expert knowledge is required, which leads to Hypothesis **H3**:

H3 CNNs enable instant cloud classification without feature engineering and selection by domain experts and include large scale spatial context.

The developed novel cloud mask generation approach *CS-CNN* uses a CNN segmentation architecture and classifies all pixels of an MSG scene simultaneously instead of individually. We showed that the architecture of CS-CNN is very well suited to process multispectral remote sensing data to classify clouds. Compared to the frequently used RF, CS-CNN requires significantly less domain expert knowledge and effort. The results of CS-CNN show higher accuracy and are produced faster. Additionally, they are significantly more robust, i.e., they are more consistent. The diurnal and seasonal, as well as the inter-class variation is very low for CS-CNN, while the RF with the highest accuracy and skill shows a larger variance. CS-CNN provides high overall accuracy (0.94) and HSS (0.90) values and requires only 25 ms of computing time for classifying an input scene on a GPU. This result validates Hypothesis **H3**.

While the processing of large spatio-temporal time series as batches is feasible, interactive visualization and analysis are challenging. The lack of time as an integral dimension in GIS, and the handling of big data, limits new insights, which are valuable for many domains like biodiversity or traffic. Web-based special-purpose systems provide visualizations with time sliders but without facilitating explorative workflows. Therefore, **H4** was formulated as follows:

H4 Interactive exploration and visualization for spatio-temporal data enables insights beyond pure event- or raster-based analytics.

The Visualization, Analysis, and Transformation (VAT) System is developed to enable data-driven research and explorative workflows for spatio-temporal data. The implemented back-end provides access to large raster time series and uses techniques for efficient and parallel raster handling, including pyramids and GPU processing. Operators provided by the back-end are available in the web-based user interface. Combining them creates explorative workflows, which are reusable for different spatial and temporal settings. Based on an explorative workflow,

we implemented a spatio-temporal visualization and processing of MSG data in VAT. Combining cloud mask and rainfall raster time series with plane trajectories enables identification of flight paths through clouds, including bad weather situations.

9.2 Outlook

After having discussed the results of this work, this section will give a brief outlook on future research plans and application potentials of the methods and information derived in this thesis.

The efficient and parallel processing of MSG data in Python is beneficial for other tasks, e.g., the training data generation for CS-CNN. Therefore, we published the processing chain as open-source software on Github[1]. We will update the implementation with further improvements, including concurrent processing. Furthermore, VAT already provides operators to handle MSG data preprocessing. The addition of parallel image histogram operators and an enhanced Python integration will enable FLS detection as a workflow. To enable data scientist to experiment with data from existing or ad-hoc workflows, a Python library will allow access to the data and operators provided by VAT.

The CNN approach of cloud classification shows excellent results. Therefore, we will test CNN architectures for other classification tasks. For example, we will examine the classification of individual cloud types, detection of precipitation areas, and the estimation of rainfall in our future work. Additional input data, such as cloud top pressure or model data, will be inspected. While the classification of complex cloud classes appears to require an adaption of the CS-CNN, deriving rainfall from MSG data is a more challenging task. MSG SEVIRI can only capture the top surfaces, and precipitation happens below the clouds. While methods to relate rainfall to data produced by SEVIRI are available [Beu+18; Mey+16], there is still room for improvements. Similar to existing approaches, we are working on training a CNN with MSG SEVIRI input against weather radar data, which provides rainfall data with high spatial and temporal resolution. Additionally, CNNs provide an interesting opportunity for transfer learning of existing methods for new satellites, e.g., MSG cloud mask to MTG.

The data preparation is the most time-consuming step for training and testing CNNs as well as other machine learning methods. Integrating the data preparation step as a workflow in VAT and providing an API for batch-wise data provi-

[1] github

sioning can improve this situation. Adaptions, e.g., to correct errors in the data, can be applied to the workflow without manually reprocessing the data. Trained models will also be integrated as ready-to-use operators. This way, all users can profit from models who are trained only once. Another task is an extension to other data sources and integration of real-time data. Applying a defined workflow for new time steps can facilitate the operationalization of methods like FLS detection.

Extension of available data is also interesting for exploration of different kinds of spatio-temporal data to spark new insights. One example is the correlation of animal tracks with weather data. VAT can visualize different kinds of data, and users can change time and location interactively. This correlates with the aim to connect the spatio-temporal data handling in VAT with event processing. This way, the information from raster data like FLS or precipitation can be extracted directly for moving objects like planes.

Appendices

References

[Ack+98] S. A. Ackerman, K. L. Strabala, W. P. Menzel, R. A. Frey, C. C. Moeller, and L. E. Gumley, "Discriminating clear sky from clouds with MODIS," *Journal of Geophysical Research*, vol. 103, no. D24, pp. 32 141–32 157, 1998.

[Ala+12] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki, "Nodb: Efficient query execution on raw data files," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ACM, 2012, pp. 241–252.

[Alb09] J. Albertz, *Einführung in die Fernerkundung: Grundlagen der Interpretation von Luft-und Satellitenbildern*. Wiss. Buchges., 2009.

[Alo+17] I. Alonso, A. B. Cambra, A. Munoz, T. Treibitz, and A. C. Murillo, "Coral-segmentation: Training dense labeling models with sparse ground truth.," in *ICCV Workshops*, 2017, pp. 2874–2882.

[Ame+04] Z. Ameur, S. Ameur, A. Adane, H. Sauvageot, and K. Bara, "Cloud classification using the textural features of Meteosat images," *International Journal of Remote Sensing*, vol. 25, no. 21, pp. 4491–4503, 2004.

[AAG03] N. Andrienko, G. Andrienko, and P. Gatalsky, "Exploratory spatio-temporal visualization: An analytical review," *Journal of Visual Languages & Computing*, vol. 14, no. 6, pp. 503–541, 2003, Visual Data Mining.

[Aut+15a] C. Authmann, C. Beilschmidt, J. Drönner, M. Mattig, and B. Seeger, "Rethinking spatial processing in data-intensive science," *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*, 2015.

[Aut+15b] C. Authmann, C. Beilschmidt, J. Drönner, M. Mattig, and B. Seeger, "VAT: A system for visualizing, analyzing and transforming spatial data in science," *Datenbank-Spektrum*, vol. 15, no. 3, pp. 175–184, 2015.

[BKC17] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[BAC17] J. E. Ball, D. T. Anderson, and C. S. Chan Sr., "Comprehensive survey of deep learning in remote sensing: Theories, tools, and challenges for the community," *Journal of Applied Remote Sensing*, vol. 11, no. 4, pp. 1–54, 2017.

[Ban+09] R. L. Bankert, C. Mitrescu, S. D. Miller, and R. H. Wade, "Comparison of GOES cloud classification algorithms employing explicit and implicit physics," *Journal of Applied Meteorology and Climatology*, vol. 48, no. 7, pp. 1411–1421, 2009.

[Bar+04] H. Bartels, E. Weigl, T. Reich, P. Lang, A. Wagner, O. Kohler, N. Gerlach, *et al.*, "Projekt radolan–routineverfahren zur online-aneichung der radarniederschlagsdaten mit hilfe von automatischen bodenniederschlagsstationen (ombrometer)," *Deutscher Wetterdienst, Hydrometeorologie*, vol. 5, 2004.

[BL12] A. Basset and W. Los, "Biodiversity e-Science: LifeWatch, the European Infrastructure on Biodiversity and Ecosystem Research," *Plant Biosystems*, vol. 146, no. 4, pp. 780–782, 2012.

[BCS16] L. Battle, R. Chang, and M. Stonebraker, "Dynamic prefetching of data tiles for interactive visualization," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16, San Francisco, California, USA: ACM, 2016, pp. 1363–1375.

[Bau+12] B. A. Baum, W. P. Menzel, R. A. Frey, D. C. Tobin, R. E. Holz, S. A. Ackerman, A. K. Heidinger, and P. Yang, "Modis cloud-top property refinements for collection 6," *Journal of applied meteorology and climatology*, vol. 51, no. 6, pp. 1145–1163, 2012.

[BM11] P. Baumann and S. Meissl, *Wcs 2.0 application profile-earth observation*, 2011.

[Bau99] P. Baumann, "A database array algebra for spatio-temporal data and beyond," in *Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems*, Jul. 1999, pp. 76–93.

[Bau18] P. Baumann, "Datacube standards and their contribution to analysis-ready data," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, 2018, pp. 2051–2053.

[Bau+98] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann, "The multidimensional database system rasdaman," *Acm Sigmod Record*, vol. 27, no. 2, pp. 575–577, 1998.

[BDM13] P. Baumann, A. M. Dumitru, and V. Merticariu, "The array database that is not a database: File based array query answering in rasdaman," in *International Symposium on Spatial and Temporal Databases*, Springer, 2013, pp. 478–483.

[BJS09] P. Baumann, C. Jucovschi, and S. Stancu-Mara, "Efficient map portrayal using a general-purpose query language," in *International Conference on Database and Expert Systems Applications*, Springer, 2009, pp. 153–163.

[Bau+16] P. Baumann, P. Mazzetti, J. Ungar, R. Barbera, D. Barboni, A. Beccati, L. Bigagli, E. Boldrini, R. Bruno, A. Calanducci, P. Campalani, O. Clements, A. Dumitru, M. Grant, P. Herzig, G. Kakaletris, J. Laxton, P. Koltsida, K. Lipskoch, A. R. Mahdiraji, S. Mantovani, V. Merticariu, A. Messina, D. Misev, S. Natali, S. Nativi, J. Oosthoek, M. Pappalardo, J. Passmore, A. P. Rossi, F. Rundo, M. Sen, V. Sorbera, D. Sullivan, M. Torrisi, L. Trovato, M. G. Veratelli, and S. Wagner, "Big data analytics for earth sciences: The earthserver approach," *International Journal of Digital Earth*, vol. 9, no. 1, pp. 3–29, 2016.

[Bau+19] P. Baumann, D. Misev, V. Merticariu, and B. P. Huu, "Datacubes: Towards space/time analysis-ready data," in *Service-Oriented Mapping*, Springer, 2019, pp. 269–299.

[Bau+18] P. Baumann, D. Misev, V. Merticariu, B. P. Huu, and B. Bell, "Rasdaman: Spatio-temporal datacubes on steroids," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '18, Seattle, Washington: ACM, 2018, pp. 604–607.

[Bea14] J. H. Beach, "Conceptualizing and Managing Paleontological Collection Data with Specify Software," in *GSA Annual Meeting: Advancing the Digitization of Paleontology and Geoscience Collections: Projects, Programs, and Practices II*, 2014.

[Bei+19a] C. Beilschmidt, J. Drönner, N. Glombiewski, C. Heigele, J. Holznigenkemper, A. Isenberg, M. Körber, M. Mattig, A. Morgen, and B. Seeger, "Pretty fly for a VAT GUI: visualizing event patterns for flight data," in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, DEBS 2019, Darmstadt, Germany, June 24-28, 2019.*, 2019, pp. 224–227.

[Bei+17a] C. Beilschmidt, J. Drönner, M. Mattig, M. Schmidt, C. Authmann, A. Niamir, T. Hickler, and B. Seeger, "Interactive data exploration for geoscience," in *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs ,,Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Workshopband*, 2017, pp. 117–126.

[Bei+17b] C. Beilschmidt, J. Drönner, M. Mattig, M. Schmidt, C. Authmann, A. Niamir, T. Hickler, and B. Seeger, "VAT: A Scientific Toolbox for Interactive Geodata Exploration," *Datenbank-Spektrum*, vol. 17, no. 3, pp. 233–243, 2017.

[Bei+17c] C. Beilschmidt, J. Drönner, M. Mattig, and B. Seeger, "VAT: A system for data-driven biodiversity research," in *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, 2017, pp. 546–549.

[Bei+19b] C. Beilschmidt, M. Mattig, T. Fober, and B. Seeger, "An efficient aggregation and overlap removal algorithm for circle maps," *GeoInformatica*, vol. 23, no. 3, pp. 473–498, 2019.

[Bel11] L. Belbin, "The Atlas of Living Australia's Spatial Portal," in *Proceedings of the Environmental Information Management Conference 2011 (EIM 2011)*, vol. 29, 2011, pp. 39–43.

[Bel15] R. E. Bellman, *Adaptive control processes: a guided tour*. Princeton university press, 2015, vol. 2045.

[Ben+17] N. Benas, S. Finkensieper, M. Stengel, G.-J. van Zadelhoff, T. Hanschmann, R. Hollmann, and J. F. Meirink, "The MSG-SEVIRI-based cloud property data record CLAAS-2," *Earth System Science Data*, vol. 9, no. 2, pp. 415–434, Jul. 2017.

[BB91] J. Bendix and M. Bachmann, "Ein operationell einsetzbares Verfahren zur Nebelerkennung auf der Basis von AVHRR-Daten der NOAA-Satelliten," *Meteorologische Rundschau*, vol. 43, no. 6, pp. 169–178, 1991.

[BEK11] J. Bendix, W. Eugster, and O. Klemm, "Fog - boon or bane?" *Erdkunde*, vol. 65, no. 3, pp. 229–232, 2011.

[Ben02] J. Bendix, "A satellite-based climatology of fog and low-level stratus in Germany and adjacent areas," *Atmospheric Research*, vol. 64, no. 1-4, pp. 3–18, 2002.

[Bes+16] K. Bessho, K. Date, M. Hayashi, A. Ikeda, T. Imai, H. Inoue, Y. Kumagai, T. Miyakawa, H. Murata, T. Ohno, *et al.*, "An introduction to himawari-8/9 - japan's new-generation geostationary meteorological satellites," *Journal of the Meteorological Society of Japan. Ser. II*, vol. 94, no. 2, pp. 151–183, 2016.

[Beu+18] L. Beusch, L. Foresti, M. Gabella, and U. Hamann, "Satellite-based rainfall retrieval: From generalized linear models to artificial neural networks," *Remote Sensing*, vol. 10, no. 6, p. 939, 2018.

[Bot16] A. Bott, *Synoptische Meteorologie: Methoden der Wetteranalyse und-prognose*. Springer-Verlag, 2016.

[Bre01] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[BEB05] L. S. Bruijnzeel, W. Eugster, and R. Burkard, "Fog as a Hydrologic Input," *Encyclopedia of Hydrological Sciences*, pp. 559–582, 2005.

[Cas+08] D. Castaño-Díez, D. Moser, A. Schoenegger, S. Pruggnaller, and A. S. Frangakis, "Performance evaluation of image processing algorithms on the gpu," *Journal of Structural Biology*, vol. 164, no. 1, pp. 153–160, 2008.

[Cas+15] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva, "Land Use Classification in Remote Sensing Images by Convolutional Neural Networks," *arXiv preprint arXiv:1508.00092*, pp. 1–11, 2015.

[Cer06] J. Cermak, "SOFOS - A new Satellite-based Operational Fog Observation Scheme," PhD thesis, Philipps-University of Marburg, 2006, p. 165.

[Cer12] J. Cermak, "Low clouds and fog along the South-Western African coast - Satellite-based retrieval and spatial patterns," *Atmospheric Research*, vol. 116, pp. 15–21, 2012.

[CB07] J. Cermak and J. Bendix, "Dynamical nighttime fog/low stratus detection based on Meteosat SEVIRI data: A feasibility study," *Pure and Applied Geophysics*, vol. 164, no. 6-7, pp. 1179–1192, 2007.

[CB08] J. Cermak and J. Bendix, "A novel approach to fog/low stratus detection using Meteosat 8 data," *Atmospheric Research*, vol. 87, no. 3-4, pp. 279–292, 2008.

[CBD08] J. Cermak, J. Bendix, and M. Dobbermann, "Fmet - an integrated framework for meteosat data processing for operational scientific applications," *Computers & Geosciences*, vol. 34, no. 11, pp. 1638–1644, 2008.

[Cer+09] J. Cermak, R. M. Eastman, J. Bendix, and S. G. Warren, "European climatology of fog and low stratus based on geostationary satellite observations," *Quarterly Journal of the Royal Meteorological Society*, vol. 135, no. 645, pp. 2125–2130, 2009.

[CC98] E. J. Chaisson and E. Chaisson, *The Hubble wars: Astrophysics meets astropolitics in the two-billion-dollar struggle over the Hubble Space Telescope*. Harvard University Press, 1998.

[CMI11] E. Christophe, J. Michel, and J. Inglada, "Remote sensing processing: From multicore to gpu," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 643–652, Sep. 2011.

[CM 16] CM SAF, "Algorithm Theoretical Basis Document SEVIRI Cloud Physical Products CLAAS Edition 2," SAF/CM/KNMI/ATBD/SEVIRI/CPP, Tech. Rep. 2.2, 2016, pp. 1–24.

[CW14] M. J. Costello and J. Wieczorek, "Best Practice for Biodiversity Data Management and Publication," *Biological Conservation*, vol. 173, pp. 68–73, 2014.

[De 13] N. De Lange, *Geoinformatik: in Theorie und Praxis*. Springer-Verlag, 2013.

[dJes+12] J. de Jesus, P. Walker, M. Grant, and S. Groom, "Wps orchestration using the taverna workbench: The escience approach," *Computers & Geosciences*, vol. 47, pp. 75–86, 2012, Towards a Geoprocessing Web.

[DGF13] M. Derrien, H. Gléau, and P. Fernandez, "Algorithm theoretical basis document for cloud products (cma-pge01 v3. 2, ct-pge02 v2. 2 & ctthpge03 v2. 2)," NWC SAF, Tech. Rep., 2013.

[DGG+14] M. Diepenbroek, F. Glöckner, P. Grobe, *et al.*, "Towards an Integrated Biodiversity and Ecological Research Data Management and Archiving Platform: The German Federation for the Curation of Biological Data (GFBio)," in *GI-Jahrestagung*, 2014, pp. 1711–1721.

[Doz89] J. Dozier, "Spectral signature of alpine snow cover from the landsat thematic mapper," *Remote Sensing of Environment*, vol. 28, no. August 1988, pp. 9–22, 1989.

[Drö+19] J. Drönner, S. Egli, B. Thies, J. Bendix, and B. Seeger, "FFLSD - fast fog and low stratus detection tool for large satellite time-series," *Computers & Geosciences*, vol. 128, pp. 51–59, 2019.

[Drö+18] J. Drönner, N. Korfhage, S. Egli, M. Mühling, B. Thies, J. Bendix, B. Freisleben, and B. Seeger, "Fast cloud segmentation using convolutional neural networks," *Remote Sensing*, vol. 10, no. 11, p. 1782, 2018.

[Dru+12] M. Drusch, U. Del Bello, S. Carlier, O. Colin, V. Fernandez, F. Gascon, B. Hoersch, C. Isola, P. Laberinti, P. Martimort, *et al.*, "Sentinel-2: Esa's optical high-resolution mission for gmes operational services," *Remote sensing of Environment*, vol. 120, pp. 25–36, 2012.

[Egl+17] S. Egli, B. Thies, J. Drönner, J. Cermak, and J. Bendix, "A 10 year fog and low stratus climatology for Europe based on Meteosat Second Generation data," *Quarterly Journal of the Royal Meteorological Society*, vol. 143, no. 702, pp. 530–541, Jan. 2017.

[ETB18] S. Egli, B. Thies, and J. Bendix, "A hybrid approach for fog retrieval based on a combination of satellite and ground truth data," *Remote Sensing*, vol. 10, no. 4, 2018.

[Erw+99] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis, "Spatio-temporal data types: An approach to modeling and querying moving objects in databases," *GeoInformatica*, vol. 3, no. 3, pp. 269–296, Sep. 1999.

[EUM10a] EUMETSAT, "High Rate SEVIRI Level 1.5 Image Data - MSG - 0 degree," European Organisation for the Exploitation of Meteorological Satellites, Darmstadt, Tech. Rep., 2010.

[EUM10b] EUMETSAT, "MSG Level 1 . 5 Image Data Format Description," European Organisation for the Exploitation of Meteorological Satellites, Darmstadt, Tech. Rep., 2010, pp. 1–127.

[EUM11] EUMETSAT, *Public wavelet transform decompression library software, version 2.06*, version 2.06, Accessed 2017-12-12, 2011.

[EUM12a] EUMETSAT, "Conversion from radiances to reflectances for SEVIRI warm channels," European Organisation for the Exploitation of Meteorological Satellites, Darmstadt, Tech. Rep., 2012.

[EUM12b] EUMETSAT, "Effective Radiance and Brightness Temperature Relation Tables for Meteosat Second Generation," European Organisation for the Exploitation of Meteorological Satellites, Darmstadt, Tech. Rep., 2012.

[EUM12c] EUMETSAT, "The Conversion from Effective Radiances to Equivalent Brightness Temperatures," European Organisation for the Exploitation of Meteorological Satellites, Darmstadt, Tech. Rep., 2012.

[EUM15] EUMETSAT, "MSG Ground Segment LRIT HRIT Mission Specific Implementation," European Organisation for the Exploitation of Meteorological Satellites, Darmstadt, Tech. Rep., 2015, pp. 1–154.

[Far+07] T. G. Farr, P. A. Rosen, E. Caro, R. Crippen, R. Duren, S. Hensley, M. Kobrick, M. Paller, E. Rodriguez, L. Roth, *et al.*, "The shuttle radar topography mission," *Reviews of geophysics*, vol. 45, no. 2, 2007.

[FH17] S. E. Fick and R. J. Hijmans, "Worldclim 2: New 1-km spatial resolution climate surfaces for global land areas," *International journal of climatology*, vol. 37, no. 12, pp. 4302–4315, 2017.

[FWC19] N. Flood, F. Watson, and L. Collett, "Using a u-net convolutional neural network to map woody vegetation extent from high resolution satellite imagery across queensland, australia," *International Journal of Applied Earth Observation and Geoinformation*, vol. 82, p. 101 897, 2019.

[Fre+08] R. A. Frey, S. A. Ackerman, Y. Liu, K. I. Strabala, H. Zhang, J. R. Key, and X. Wang, "Cloud detection with modis. part i: Improvements in the modis cloud mask for collection 5," *Journal of Atmospheric and Oceanic Technology*, vol. 25, no. 7, pp. 1057–1072, 2008.

[Gan+11] G. Ganci, A. Vicari, S. Bonfiglio, G. Gallo, and C. del Negro, "A texton-based cloud detection algorithm for MSG-SEVIRI multispectral images," *Geomatics, Natural Hazards and Risk*, vol. 2, no. 3, pp. 279–290, 2011.

[Gas+13] B. R. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, "OpenCL Case Study: Histogram," in *Heterogeneous Computing with OpenCL*, Elsevier, 2013, pp. 183–194.

[Gas+11] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[GDA17] GDAL Development Team, *Gdal - geospatial data abstraction library, version 2.1*, Accessed 2017-12-12, Open Source Geospatial Foundation, 2017.

[GLP19] S. Gebbert, T. Leppelt, and E. Pebesma, "A topology based spatiotemporal map algebra for big data analysis," *Data*, vol. 4, no. 2, 2019.

[Gia15] R. Giachetta, "A framework for processing large scale geospatial and remote sensing data in mapreduce environment," *Computers & Graphics*, vol. 49, pp. 37–46, 2015.

[Gil+18] Y. Gil, S. A. Pierce, H. Babaie, A. Banerjee, K. Borne, G. Bust, M. Cheatham, I. Ebert-Uphoff, C. Gomes, M. Hill, *et al.*, "Intelligent systems for geosciences: An essential research agenda," *Communications of the ACM*, vol. 62, no. 1, pp. 76–84, 2018.

[GWM08] R. Gloaguen, A. R. Wijaya, and P. R. Marpu, "Geostatistical texture classification of tropical rainforest in indonesia," in *Quality Aspect in Spatial Data Mining*, 2008.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[Gor+17] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, "Google earth engine: Planetary-scale geospatial analysis for everyone," *Remote Sensing of Environment*, vol. 202, pp. 18–27, 2017.

[GO15] A. Graser and V. Olaya, "Processing: A python framework for the seamless integration of geoprocessing tools in qgis," *ISPRS International Journal of Geo-Information*, vol. 4, no. 4, pp. 2219–2245, 2015.

[Gri15] R. E. Griffin, "When are Old Data New Data?" *GeoResJ*, vol. 6, pp. 92–97, 2015.

[Gu+89] Z. Gu, C. Duncan, E. Renshaw, M. Mugglestone, C. Cowan, and P. Grant, "Comparison of techniques for measuring cloud texture in remotely sensed satellite meteorological image data," *IEE Proceedings F Radar and Signal Processing*, vol. 136, no. 5, p. 236, 1989.

[Gul+07] I. Gultepe, R. Tardif, S. C. Michaelides, J. Cermak, A. Bott, J. Bendix, M. D. Müller, M. Pagowski, B. Hansen, G. Ellrod, W. Jacobs, G. Toth, and S. G. Cober, "Fog research: A review of past achievements and future perspectives," *Pure and Applied Geophysics*, vol. 164, no. 6, pp. 1121–1159, 2007.

[Güt+05] R. H. Güting, V. Almeida, D. Ansorge, T. Behr, Z. Ding, T. Hose, F. Hoffmann, M. Spiekermann, and U. Telle, "Secondo: An extensible dbms platform for research prototyping and teaching," in *21st International Conference on Data Engineering (ICDE'05)*, IEEE, 2005, pp. 1115–1116.

[Güt+00] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis, "A foundation for representing and querying moving objects," *ACM Trans. Database Syst.*, vol. 25, no. 1, pp. 1–42, Mar. 2000.

[HBS19] S. Hagedorn, O. Birli, and K.-U. Sattler, "Processing large raster and vector data in apache spark," in *BTW 2019*, T. Grust, F. Naumann, A. Böhm, W. Lehner, T. Härder, E. Rahm, A. Heuer, M. Klettke, and H. Meyer, Eds., Gesellschaft für Informatik, Bonn, 2019, pp. 551–554.

[Han+03] M. C. Hansen, R. S. DeFries, J. R. G. Townshend, M. Carroll, C. Dimiceli, and R. A. Sohlberg, "Global percent tree cover at a spatial resolution of 500 meters: First results of the modis vegetation continuous fields algorithm," *Earth Interactions*, vol. 7, no. 10, pp. 1–15, 2003.

[Han+08] M. C. Hansen, D. P. Roy, E. Lindquist, B. Adusei, C. O. Justice, and A. Altstatt, "A method for integrating modis and landsat data for systematic monitoring of forest cover and change in the congo basin," *Remote Sensing of Environment*, vol. 112, no. 5, pp. 2495–2513, 2008, Earth Observations for Terrestrial Biodiversity and Ecosystems Special Issue.

[Har06] Q. J. Hart, "GeoStreams: An online geospatial image database," PhD thesis, UNIVERSITY OF CALIFORNIA DAVIS, 2006.

[He+17] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2017, pp. 2980–2988.

[HJP13] M. Heistermann, S. Jacobi, and T. Pfaff, "An open source library for processing weather radar data (wradlib)," *Hydrology and Earth System Sciences*, vol. 17, no. 2, pp. 863–871, 2013.

[Hir+17] J. N. Hird, E. R. DeLancey, G. J. McDermid, and J. Kariyeva, "Google earth engine, open-access satellite data, and machine learning in support of large-area probabilistic wetland mapping," *Remote Sensing*, vol. 9, no. 12, 2017.

[HFS10] J. Hocking, P. N. Francis, and R. Saunders, "Cloud detection in Meteosat Second Generation imagery at the Met Office," *Meteorological Applications*, vol. 18, no. 3, 2010.

[Hol+08] R. Holz, S. Ackerman, F. Nagle, R. Frey, S. Dutcher, R. Kuehn, M. Vaughan, and B. Baum, "Global moderate resolution imaging spectroradiometer (modis) cloud detection and height evaluation using caliop," *Journal of Geophysical Research: Atmospheres*, vol. 113, no. D8, 2008.

[Hoß+13] B. Hoßbach, N. Glombiewski, A. Morgen, F. Ritter, and B. Seeger, "JEPC: the java event processing connectivity," *Datenbank-Spektrum*, vol. 13, no. 3, pp. 167–178, 2013.

[Hou14a] R. A. Houze, "Chapter 8 - cumulonimbus and severe storms," in *Cloud Dynamics*, ser. International Geophysics, R. A. Houze, Ed., vol. 104, Academic Press, 2014, pp. 187–236.

[Hou14b] R. A. Houze, *Cloud dynamics*, R. A. Houze, Ed., ser. International Geophysics. Academic press, 2014, vol. 104.

[Hu+18a] F. Hu, M. Xu, J. Yang, Y. Liang, K. Cui, M. M. Little, C. S. Lynnes, D. Q. Duffy, and C. Yang, "Evaluating the open source data containers for handling big geospatial raster data," *ISPRS International Journal of Geo-Information*, vol. 7, no. 4, 2018.

[Hu+18b] F. Hu, C. Yang, J. L. Schnase, D. Q. Duffy, M. Xu, M. K. Bowen, T. Lee, and W. Song, "Climatespark: An in-memory distributed computing framework for big climate data analytics," *Computers & geosciences*, vol. 115, pp. 154–166, 2018.

[Hua+16] F. Huang, J. Zhou, J. Tao, X. Tan, S. Liang, and J. Cheng, "Pmodtran: A parallel implementation based on modtran for massive remote sensing data processing," *International journal of digital earth*, vol. 9, no. 9, pp. 819–834, 2016.

[HdB09] O. Huisman and R. A. de By, "Principles of Geographic Information Systems," *ITC Educational Textbook Series*, vol. 1, 2009.

[Hun73] G. E. Hunt, "Radiative properties of terrestrial clouds at visible and infra-red thermal window wavelengths," *Quarterly Journal of the Royal Meteorological Society*, vol. 99, no. 420, pp. 346–369, 1973.

[IW09] R. Ikeda and J. Widom, "Data lineage: A survey," Stanford University, Technical Report, 2009.

[IC09] J. Inglada and E. Christophe, "The orfeo toolbox remote sensing image processing software," in *2009 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, vol. 4, 2009, pp. IV–733.

[Jää+15] P. Jääskeläinen, C. S. de La Lama, E. Schnetter, K. Raiskila, J. Takala, and H. Berg, "Pocl: A performance-portable opencl implementation," *International Journal of Parallel Programming*, vol. 43, no. 5, pp. 752–785, 2015.

[JMG12] W. Jetz, J. M. McPherson, and R. P. Guralnick, "Integrating biodiversity distribution knowledge: Toward a global map of life," *Trends in ecology & evolution*, vol. 27, no. 3, pp. 151–159, 2012.

[Jia+14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, ACM, 2014, pp. 675–678.

[JS03] I. T. Jolliffe and D. B. Stephenson, *Forecast verification: a practitioner's guide in atmospheric science*. John Wiley & Sons, 2003.

[Kar+17] K.-G. Karlsson, K. Anttila, J. Trentmann, M. Stengel, J. Fokke Meirink, A. Devasthale, T. Hanschmann, S. Kothe, E. Jääskeläinen, J. Sedlar, N. Benas, G.-J. van Zadelhoff, C. Schlundt, D. Stein, S. Finkensieper, N. Håkansson, and R. Hollmann, "Clara-a2: The second edition of the cm saf cloud and radiation data record from 34 years of global avhrr data," *Atmospheric Chemistry and Physics*, vol. 17, no. 9, pp. 5809–5828, 2017.

[Kay+15] R. Kays, M. C. Crofoot, W. Jetz, and M. Wikelski, "Terrestrial animal tracking as an eye on life and planet," *Science*, vol. 348, no. 6240, 2015.

[Khr12] Khronos OpenCL Working Group, *The opencl specification, version 1.2*, (15.07.2019), 2012.

[KB14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv e-prints*, arXiv:1412.6980, arXiv:1412.6980, Dec. 2014.

[KL16] O. Klemm and N.-H. Lin, "What Causes Observed Fog Trends: Air Quality or Climate Change?" *Aerosol and Air Quality Research*, vol. 16, no. 5, pp. 1131–1142, 2016.

[Klö+12] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "Pycuda and pyopencl: A scripting-based approach to gpu run-time code generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[Köh+17] C. Köhler, A. Steiner, Y.-M. Saint-Drenan, D. Ernst, A. Bergmann-Dick, M. Zirkelbach, Z. Ben Bouallègue, I. Metzinger, and B. Ritter, "Critical weather situations for renewable energies – Part B: Low stratus risk for solar power," *Renewable Energy*, vol. 101, pp. 794–803, Feb. 2017.

[Kör+19] M. Körber, N. Glombiewski, A. Morgen, and B. Seeger, "Tpstream: Low-latency and high-throughput temporal pattern matching on event streams," *Distributed and Parallel Databases*, Jul. 2019.

[Krä07] J. Krämer, "Continuous queries over data stream - semantics and implementation," PhD thesis, University of Marburg, Germany, 2007.

[KS05] J. Krämer and B. Seeger, "A temporal foundation for continuous queries over data streams," in *Advances in Data Management 2005, Proceedings of the Eleventh International Conference on Management of Data, January 6, 7, and 8, 2005, Goa, India*, 2005, pp. 70–82.

[Kra+11] B. Kranstauber, A. Cameron, R. Weinzerl, T. Fountain, S. Tilak, M. Wikelski, and R. Kays, "The movebank data model for animal tracking," *Environmental Modelling & Software*, vol. 26, no. 6, pp. 834–835, 2011.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, pp. 1–9, 2012.

[Kue+14] C. Kuenzer, M. Ottinger, M. Wegmann, H. Guo, C. Wang, J. Zhang, S. Dech, and M. Wikelski, "Earth observation satellite sensors for biodiversity monitoring: Potentials and bottlenecks," *International Journal of Remote Sensing*, vol. 35, no. 18, pp. 6599–6647, 2014.

[Küh+14] M. Kühnlein, T. Appelhans, B. Thies, and T. Nauss, "Improving the accuracy of rainfall rates from optical satellite sensors with machine learning — a random forests-based approach applied to msg seviri," *Remote Sensing of Environment*, vol. 141, pp. 129–143, 2014.

[LPS17] S. Ladra, J. R. Paramá, and F. Silva-Coira, "Scalable and queryable compressed storage structure for raster data," *Information Systems*, vol. 72, pp. 179–204, 2017.

[LA04] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, IEEE Computer Society, 2004, p. 75.

[LA18] M. Lazri and S. Ameur, "Combination of support vector machine, artificial neural network and random forest for improving the classification of convective and stratiform rain using spectral features of seviri data," *Atmospheric Research*, vol. 203, pp. 118–129, 2018.

[Le +17] M. Le Goff, J. Tourneret, H. Wendt, M. Ortner, and M. Spigai, "Deep learning for cloud detection," in *8th International Conference of Pattern Recognition Systems (ICPRS 2017)*, Jul. 2017, pp. 1–6.

[LBH15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[LWA04] Y. Lee, G. Wahba, and S. A. Ackerman, "Cloud Classification of Satellite Radiance Data by Multicategory Support Vector Machines," *Journal of Atmospheric and Oceanic Technology*, vol. 21, no. 2, pp. 159–169, Feb. 2004.

[LT15] F. Li and G. Taylor, "Alter-CNN: An approach to learning from label proportions with application to ice-water classification," in *Neural Information Processing Systems 28 (NIPS) Deep Learning and Representation Learning Workshop on Learning and Privacy with Incomplete Data and Weak Supervision*, 2015.

[Li+19] Z. Li, H. Shen, Q. Cheng, Y. Liu, S. You, and Z. He, "Deep learning based cloud detection for medium and high resolution remote sensing images of different sensors," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 150, pp. 197–212, 2019.

[Liu+19] C.-C. Liu, Y.-C. Zhang, P.-Y. Chen, C.-C. Lai, Y.-H. Chen, J.-H. Cheng, and M.-H. Ko, "Clouds classification from sentinel-2 imagery with deep residual learning and semantic image segmentation," *Remote Sensing*, vol. 11, no. 2, 2019.

[Liu+14] F. Liu, K. Lee, I. Roy, V. Talwar, S. Chen, J. Chang, and P. Ranganathan, "Gpu accelerated array queries: The good, the bad, and the promising," 2014.

[Löf+17] F. Löffler, K. Opasjumruskit, N. Karam, D. Fichtmüller, U. Schindler, F. Klan, C. Müller-Birn, and M. Diepenbroek, "Honey Bee Versus Apis Mellifera: A Semantic Search for Biological Data," in *The Semantic Web: ESWC 2017 Satellite Events*, Cham, Switzerland: Sprin-ger International Publishing, 2017, pp. 98–103.

[LSD15] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[Ma+15] Y. Ma, H. Wu, L. Wang, B. Huang, R. Ranjan, A. Zomaya, and W. Jie, "Remote sensing big data computing: Challenges and opportunities," *Future Generation Computer Systems*, vol. 51, pp. 47–60, 2015.

[Mag+17] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Convolutional Neural Networks for Large-Scale Remote-Sensing Image Classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 645–657, 2017.

[Mar14] S. Marsland, *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2014.

[Men+18] W. P. Menzel, T. J. Schmit, P. Zhang, and J. Li, "Satellite-based atmospheric infrared sounder development and applications," *Bulletin of the American Meteorological Society*, vol. 99, no. 3, pp. 583–603, 2018.

[MDN17] H. Meyer, J. Drönner, and T. Nauss, "Satellite-based high-resolution mapping of rainfall over southern africa," *Atmospheric Measurement Techniques*, vol. 10, no. 6, pp. 2009–2019, 2017.

[Mey+16] H. Meyer, M. Kühnlein, T. Appelhans, and T. Nauss, "Comparison of four machine learning algorithms for their applicability in satellite-based optical rainfall retrievals," *Atmospheric Research*, vol. 169, pp. 424–433, Mar. 2016.

[Mey+17] H. Meyer, M. Kühnlein, C. Reudenbach, and T. Nauss, "Revealing the potential of spectral and textural predictor variables in a neural network-based rainfall retrieval technique," *Remote Sensing Letters*, vol. 8, no. 7, pp. 647–656, 2017.

[MG15] H. J. Miller and M. F. Goodchild, "Data-driven geography," *GeoJournal*, vol. 80, no. 4, pp. 449–461, Aug. 2015.

[Mus+14] J. Musial, F. Hüsler, M. Sütterlin, C. Neuhaus, and S. Wunderle, "Daytime Low Stratiform Cloud Detection on AVHRR Imagery," *Remote Sensing*, vol. 6, no. 6, pp. 5124–5150, 2014.

[NHN01] B. Nemery, P. H. M. Hoet, and A. Nemmar, "The Meuse Valley fog of 1930: an air pollution disaster," *The Lancet*, vol. 357, no. 9257, pp. 704–708, Feb. 2001.

[NM13] M. Neteler and H. Mitasova, *Open source GIS: a GRASS GIS approach*. Springer Science & Business Media, 2013, vol. 689.

[NHH15] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1520–1528.

[Nor+16] J. R. Norris, R. J. Allen, A. T. Evan, M. D. Zelinka, C. W. O'dell, and S. A. Klein, "Evidence for climate change in the satellite cloud record," *Nature*, vol. 536, no. 7614, pp. 72–75, 2016.

[NWC13] NWC SAF, "Algorithm theoretical basis document for "cloud products" (cma-pge01 v3.2, ct-pge02 v2.2 & ctth-pge03 v2.2)," SAF/NWC/CDOP/MFL/SCI/ATBD/01, Tech. Rep., 2013.

[OH15] R. O. Obe and L. S. Hsu, *PostGIS in Action*. Manning Publications Co., 2015.

[Ope06] Open Geospatial Consortium, "Web Map Server (WMS) Implementation Specification," *OpenGIS Project Document*, 2006.

[Ope08] Open Geospatial Consortium, "Web Coverage Service (WCS) Implementation Standard," *OpenGIS Project Document*, 2008.

[Ope10a] Open Geospatial Consortium, "OpenGIS Implementation Standard for Geographic Information - Simple Feature Access," *OpenGIS Project Document*, 2010.

[Ope10b] Open Geospatial Consortium, "OpenGIS Web Feature Service (WFS) 2.0 Interface Standard," *OpenGIS Project Document*, 2010.

[Ove+11] J. T. Overpeck, G. A. Meehl, S. Bony, and D. R. Easterling, "Climate data challenges in the 21st century," *Science*, vol. 331, no. 6018, pp. 700–702, 2011.

[Owe+08] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[PWK06] C. L. Parkinson, A. Ward, and M. D. King, "Earth science reference handbook: A guide to nasa's earth science program and earth observing satellite missions," National Aeronautics and Space Administration, Washington, D.C., Tech. Rep., 2006, p. 277.

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[Pel+04] N. Pelekis, B. Theodulidis, I. Kopanakis, and Y. Teodoridis, "Literature review of spatio-temporal database models," *The Knowledge Engineering Review*, vol. 19, no. 3, pp. 235–274, 2004.

[PWP18] C. Pelletier, G. I. Webb, and F. Petitjean, "Temporal convolutional neural network for the classification of satellite image time series," *CoRR*, vol. abs/1811.10166, 2018.

[Per+16] S. Peronaci, A. Taravat, F. D. Frate, and N. Oppelt, "Use of narx neural networks for meteosat second generation seviri very short-term cloud mask forecasting," *International Journal of Remote Sensing*, vol. 37, no. 24, pp. 6205–6215, 2016.

[Peu99] D. J. Peuquet, "Time in gis and geographical databases," *Geographical information systems*, vol. 1, pp. 91–103, 1999.

[PSF12] G. Planthaber, M. Stonebraker, and J. Frew, "Earthdb: Scalable analysis of modis data using scidb," in *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, ser. BigSpatial '12, Redondo Beach, California: ACM, 2012, pp. 11–19.

[Ras+18] M. Raspaud, D. Hoese, A. Dybbroe, P. Lahtinen, A. Devasthale, M. Itkin, U. Hamann, L. Ø. Rasmussen, E. S. Nielsen, T. Leppelt, A. Maul, C. Kliche, and H. Thorsteinsson, "Pytroll: An open-source, community-driven python framework to process earth observation satellite data," *Bulletin of the American Meteorological Society*, vol. 99, no. 7, pp. 1329–1336, 2018.

[Raz12] A. Raza, "Working with spatio-temporal data type," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 39, B2, 2012.

[RR99] J. A. Richards and J. Richards, *Remote sensing digital image analysis*. Springer, 1999, vol. 3.

[RWD90] G. X. Ritter, J. N. Wilson, and J. L. Davidson, "Image algebra: An overview," *Computer Vision, Graphics, and Image Processing*, vol. 49, no. 3, pp. 297–331, 1990.

[ROC18] ROCm Development Team, *Radeon open compute platform, version 1.9.1*, Accessed 2018-12-12, AMD Corporation, 2018.

[Rod19] R. A. Rodriges Zalipynis, "Chronosdb in action: Manage, process, and visualize big geospatial arrays in the cloud," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19, Amsterdam, Netherlands: ACM, 2019, pp. 1985–1988.

[Rod+09] A. Rodriguez, R. Stuhlmann, S. Tjemkes, D. Aminou, H. Stark, and P. Blythe, *Meteosat third generation: Mission and system concepts*, 2009.

[Rog+10] J. Rogers, R. Simakov, E. Soroush, P. Velikhov, M. Balazinska, D. DeWitt, B. Heath, D. Maier, S. Madden, J. Patel, M. Stonebraker, S. Zdonik, A. Smirnov, K. Knizhnik, and P. Brown, "Overview of scidb: Large scale array storage, processing and analysis," English (US), in *Proceedings of the 2010 International Conference on Management of Data, SIGMOD '10*, Jul. 2010, pp. 963–968.

[Roj13] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[RPB15] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351, (available on arXiv:1505.04597 [cs.CV]), Springer, 2015, pp. 234–241.

[Rot13] R. E. Roth, "Interactive Maps: What we know and what we need to know," *Journal of Spatial Information Science*, vol. 6, no. 6, pp. 59–115, 2013.

[RC13] F. Rusu and Y. Cheng, "A survey on array storage, query languages, and systems," *CoRR*, vol. abs/1302.0103, 2013.

[Sam90] H. Samet, *Applications of spatial data structures - computer graphics, image processing, and GIS*. Addison-Wesley, 1990.

[Sán+15] S. Sánchez, R. Ramalho, L. Sousa, and A. Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on gpus," *Journal of Real-Time Image Processing*, vol. 10, no. 3, pp. 469–483, Sep. 2015.

[SK88] R. W. Saunders and K. T. Kriebel, "An improved method for detecting clear sky and cloudy radiances from AVHRR data," *International Journal of Remote Sensing*, vol. 9, no. 1, pp. 123–150, 1988.

[Sch+14] M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, and M. Wilhelm, "Bringing up opensky: A large-scale ads-b sensor network for research," in *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '14, Berlin, Germany: IEEE Press, 2014, pp. 83–94.

[SMM04] C. Schillings, H. Mannstein, and R. Meyer, "Operational method for deriving high resolution direct normal irradiance from satellite data," *Solar Energy*, vol. 76, no. 4, pp. 475–484, Apr. 2004.

[Sch+02] J. Schmetz, P. Pili, S. Tjemkes, D. Just, J. Kerkmann, S. Rota, and A. Ratier, "An introduction to Meteosat Second Generation (MSG)," *Bulletin of the American Meteorological Society*, vol. 83, no. 7, pp. 977–992, 2002.

[Sch+05] T. J. Schmit, M. M. Gunshor, W. P. Menzel, J. J. Gurka, J. Li, and A. S. Bachmeier, "Introducing the next-generation advanced baseline imager on goes-r," *Bulletin of the American Meteorological Society*, vol. 86, no. 8, pp. 1079–1096, 2005.

[SRE12] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri, "Nih image to imagej: 25 years of image analysis," *Nature methods*, vol. 9, no. 7, p. 671, 2012.

[Sch+15] H. Schulz, B. Waldvogel, R. Sheikh, and S. Behnke, "Curfil: Random forests for image labeling on gpu.," in *VISAPP (2)*, 2015, pp. 156–164.

[Sch+16] H. M. Schulz, B. Thies, S.-C. Chang, and J. Bendix, "Detection of ground fog in mountainous areas from modis (collection 051) daytime data using a statistical approach.," *Atmospheric Measurement Techniques*, vol. 9, no. 3, 2016.

[SJ10] L. M. Scott and M. V. Janikas, "Spatial statistics in arcgis," in *Handbook of applied spatial analysis*, Springer, 2010, pp. 27–41.

[SW04] B. Seeger and P. Widmayer, "Geographic Information Systems," in *Handbook of Data Structures and Applications*, D. P. Mehta and S. Sahni, Eds., Chapman and Hall/CRC, 2004, pp. 55–1–55–22.

[SS17] M. Seidemann and B. Seeger, "Chronicledb: A high-performance event store," in *EDBT 201*, 2017, pp. 144–155.

[SB14] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[She+17] A. Shelestov, M. Lavreniuk, N. Kussul, A. Novikov, and S. Skakun, "Large scale crop classification using google earth engine platform," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Jul. 2017, pp. 3696–3699.

[Shn03] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The Craft of Information Visualization*, ser. Interactive Technologies, B. B. BEDERSON and B. SHNEIDERMAN, Eds., San Francisco: Morgan Kaufmann, 2003, pp. 364–371.

[SPC18] N. Sidhu, E. Pebesma, and G. Câmara, "Using google earth engine to detect land cover change: Singapore as a use case," *European Journal of Remote Sensing*, vol. 51, no. 1, pp. 486–500, 2018.

[Sor+15] E. Soroush, M. Balazinska, S. Krughoff, and A. Connolly, "Efficient iterative processing in the scidb parallel array engine," in *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '15, La Jolla, California: ACM, 2015, 39:1–39:6.

[SBW11] E. Soroush, M. Balazinska, and D. Wang, "Arraystore: A storage manager for complex parallel array processing," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, ACM, 2011, pp. 253–264.

[Ste+13] C. A. Steed, D. M. Ricciuto, G. Shipman, B. Smith, P. E. Thornton, D. Wang, X. Shi, and D. N. Williams, "Big Data Visual Analytics for Exploratory Earth System Simulation Analysis," *Computers & Geosciences*, vol. 61, pp. 71–82, 2013.

[Ste+17] M. Stengel, S. Stapelberg, O. Sus, C. Schlundt, C. Poulsen, G. Thomas, M. Christensen, C. Carbajal Henken, R. Preusker, J. Fischer, *et al.*, "Cloud property datasets retrieved from avhrr, modis, aatsr and meris in the framework of the cloud_cci project," *Earth System Science Data*, vol. 9, no. 2, pp. 881–904, 2017.

[Ste+08] G. L. Stephens, D. G. Vane, S. Tanelli, E. Im, S. Durden, M. Rokey, D. Reinke, P. Partain, G. G. Mace, R. Austin, T. LEcuyer, J. Haynes, M. Lebsock, K. Suzuki, D. Waliser, D. Wu, J. Kay, A. Gettelman, Z. Wang, and R. Marchand, "Cloudsat mission: Performance and early science after the first year of operation," *Journal of Geophysical Research: Atmospheres*, vol. 113, no. D8, 2008.

[SGS10] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[Sto+11] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman, "The architecture of scidb," in *International Conference on Scientific and Statistical Database Management*, Springer, 2011, pp. 1–16.

[SK91] M. Stonebraker and G. Kemnitz, "The postgres next generation database management system," *Communications of the ACM*, vol. 34, no. 10, pp. 78–92, 1991.

[SAM94] K. I. Strabala, S. A. Ackerman, and W. P. Menzel, "Cloud Properties inferred from 8-12-μm Data," *Journal of Applied Meteorologyed Meteorology*, vol. 33, no. 2, pp. 212–229, 1994.

[Stu+13] C. J. Stubenrauch, W. B. Rossow, S. Kinne, S. Ackerman, G. Cesana, H. Chepfer, L. Di Girolamo, B. Getzewich, A. Guignard, A. Heidinger, B. C. Maddux, W. P. Menzel, P. Minnis, C. Pearl, S. Platnick, C. Poulsen, J. Riedi, S. Sun-Mack, A. Walther, D. Winker, S. Zeng, and G. Zhao, "Assessment of Global Cloud Datasets from Satellites: Project and Database Initiated by the GEWEX Radiation Panel," *Bulletin of the American Meteorological Society*, vol. 94, no. 7, pp. 1031–1049, Jul. 2013.

[Stu+05] R. Stuhlmann, A. Rodriguez, S. Tjemkes, J. Grandell, A. Arriaga, J.-L. Bézy, D. Aminou, and P. Bensi, "Plans for eumetsat's third generation meteosat geostationary satellite programme," *Advances in Space Research*, vol. 36, no. 5, pp. 975–981, 2005.

[Sze10] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[TC13] R. Tapakis and A. G. Charalambides, *Equipment and methodologies for cloud detection and classification: A review*, 2013.

[Tar+15] A. Taravat, S. Proud, S. Peronaci, F. Del Frate, and N. Oppelt, "Multilayer Perceptron Neural Networks Model for Meteosat Second Generation SEVIRI Daytime Cloud Masking," *Remote Sensing*, vol. 7, no. 2, pp. 1529–1539, Feb. 2015.

[TB11] B. Thies and J. Bendix, "Satellite based remote sensing of weather and climate: Recent achievements and future perspectives," *Meteorological Applications*, vol. 18, no. 3, pp. 262–295, 2011.

[Tur+03] W. Turner, S. Spector, N. Gardiner, M. Fladeland, E. Sterling, and M. Steininger, "Remote sensing for biodiversity science and conservation," *Trends in Ecology & Evolution*, vol. 18, no. 6, pp. 306–314, 2003.

[vdWCV11] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[VYvO09] R. Vautard, P. Yiou, and G. J. van Oldenborgh, "Decline of fog, mist and haze in Europe over the past 30 years," *Nature Geoscience*, vol. 2, no. 2, pp. 115–119, 2009.

[VB99] R. Von Glasow and A. Bott, "Interaction of radiation fog with tall vegetation," *Atmospheric Environment*, vol. 33, no. 9, pp. 1333–1346, 1999.

[WNA14] Y. Wang, A. Nandi, and G. Agrawal, "Saga: Array storage as a db with support for structural aggregations," in *SSDBM*, 2014.

[Wet16] D. Wetterdienst, *Wx-radardaten*, 2016.

[WSS98] M. Wiegner, P. Seifert, and P. Schlüssel, "Radiative effect of cirrus clouds in Meteosat Second Generation Spinning Enhanced Visible and Infrared Imager channels," *Journal of geophysical research*, vol. 103, no. D18, pp. 23 217–23 230, 1998.

[WP92] B. A. Wielicki and L. Parker, "On the determination of cloud cover from satellite sensors: The effect of sensor spatial resolution," *Journal of Geophysical Research: Atmospheres*, vol. 97, no. D12, pp. 12 799–12 823, 1992.

[WJ16] A. M. Wilson and W. Jetz, "Remotely sensed high-resolution global cloud dynamics for predicting ecosystem and biodiversity distributions," *PLoS biology*, vol. 14, no. 3, pp. 1–20, Mar. 2016.

[Wol+13] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud," *Nucleic Acids Research*, vol. 41, no. Web Server issue, W557–W561, 2013.

[Wu+18] G. Wu, X. Shao, Z. Guo, Q. Chen, W. Yuan, X. Shi, Y. Xu, and R. Shibasaki, "Automatic Building Segmentation of Aerial Imagery Using Multi-Constraint Fully Convolutional Networks," *Remote Sensing*, vol. 10, no. 3, p. 407, 2018.

[Xie+17] F. Xie, M. Shi, Z. Shi, J. Yin, and D. Zhao, "Multilevel Cloud Detection in Remote Sensing Images Based on Deep Learning," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 8, pp. 3631–3640, Aug. 2017.

[Xin+18] H. Xing, S. Floratos, S. Blanas, S. Byna, M. Prabhat, K. Wu, and P. Brown, "Arraybridge: Interweaving declarative array processing in scidb with imperative hdf5-based programs," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, Apr. 2018, pp. 977–988.

[Xu+18] Y. Xu, L. Wu, Z. Xie, and Z. Chen, "Building extraction in very high resolution remote sensing imagery using deep learning and guided filters," *Remote Sensing*, vol. 10, no. 1, 2018.

[Yan+17] C. Yang, Q. Huang, Z. Li, K. Liu, and F. Hu, "Big data and cloud computing: Innovation opportunities and challenges," *International Journal of Digital Earth*, vol. 10, no. 1, pp. 13–53, 2017.

[Zah+16] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.

[Zal18] R. A. R. Zalipynis, "Chronosdb: Distributed, file based, geospatial array dbms," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1247–1261, 2018.

[ZYG15] J. Zhang, S. You, and L. Gruenwald, "Large-scale spatial data processing on gpus and gpu-accelerated clusters," *SIGSPATIAL Special*, vol. 6, no. 3, pp. 27–34, Apr. 2015.

[ZYG17] J. Zhang, S. You, and L. Gruenwald, "Towards gpu-accelerated web-gis for query-driven visual exploration," in *International Symposium on Web and Wireless Geographical Information Systems*, Springer, 2017, pp. 119–136.

[Zur+17] R. Zurita-Milla, R. Goncalves, E. Izquierdo-Verdiguier, and F. Ostermann, "Exploring vegetation phenology at continental scales: Linking temperature-based indices and land surface phenological metrics," in *Proceedings of the 2017 Conference on Big Data from Space, Toulouse, France*, 2017, pp. 28–30.

List of Figures

List of Tables

List of Abbreviations

ADAM	Adaptive Moment Estimation
ADS-B	Automatic Dependent Surveillance Broadcast
AVHRR	Advanced Very High Resolution Radiometer
CALIOP	Cloud-Aerosol Lidar with Orthogonal Polarization
CEP	Complex Event Processing
CGMS	Cordination Group of Meteorological Satellites
CLAAS-2	CLoud property dAtAset using SEVIRI - Edition 2
CM-SAF	Satellite Application Facility on Cli-mate Monitoring
CMa	Cloud Mask
CNN	Convolutional Neural Networks
CP	clear pixels
CQL	Common Query Language
CS-CNN	Cloud Segmentation CNN
CU	Compute Unit
deconv	de-convolution
ESA	European Space Agency
ESD	Earth Sun Distance
EUMETSAT	European Organisation for the Exploitation of Meteorological Satellites
FLS	Fog and Low Stratus
GBIF	Global Biodiversity Information Facility
GDAL	Geospatial Data Abstraction Library
GEE	Google Earth Engine
GEO	GEOstationary satellite
GEOS	GEOstationary Satellite view
GeoTIFF	Geo Tagged Image File Format
GFBio	German Federation for Biological data
GIS	Geographical Information Systems
GMST	Greenwich Mean Sidereal Time
GOES	Geostationary Operational Environmental Satellite

GPGPU	General Purpose GPU
GPU	Graphics Processing Unit
HDD	Hard disk drive
HDF	Hierarchical Data Format
HRIT	High Rate Information Transmission
HRV	High Resolution Visible
IR	Infrarot
LD	large droplet
LEO	Low Earth Orbiter
LIDAR	LIght Detection And Ranging
MAPPING	Marburg's Analysis, Processing and Provenance of Information for Networked Geographics
MDD	Multi-dimensional Discrete Data
METAR	Meteorological Aviation Routine Weather Reports
ML	Machine Learning
MLP	Multi Layer Perceptron
MODIS	Moderate Resolution Imaging Spectroradiometer
MSG	Meteosat Second Generation
MTG	Meteosat Third Generation
NDSI	Normalized Difference Snow Index
netCDF	Network Common Data Form
NIR	Near InfraRed
NVMe	non-volatile memory express
OGC	Open Geospatial Consortium
OGR	Open GIS Simple Feature Reference
OpenCL	Open Compute Language
OpenGL	Open Graphics Library
PCV	Pseudo Cross Variogram
POCL	Portable Computing Language
RADAR	RAdio Detection And Ranging
RDD	Resilient Distributed Dataset
ReLU	rectified linear unit
RF	Random Forest

RGB	red, green, and blue
ROD	Rodogram
SD	small droplet
SDP	Small Drop Proxy test
SDR	Software Defined Radio
SEVIRI	Spinning Enhanced Visible and InfraRed Imager
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
SRTM	Shuttle Radar Topography Mission
SSP	Sub Satellite Point
SVA	Satellite View Angle
SVM	Support Vector Machine
SZA	Solar Zenith Angle
UML	Unified Modeling Language
VAT	Visualization, Analysis, and Transformation System
VIS	Visible
WAVE	Workflow, Analysis and Visualization Editor
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WorldClim	WorldClim - Global Climate Data
WV	Water Vapor